A

forth

PROGRAMOZÁSI NYELV

Seres—Fenyő—Gyalogh

MK

Seres Attila–Fenyő László–Gyalogh Kálmán

A FORTH programozási nyelv

Műszaki Könyvkiadó, Budapest, 1986

Lektorálta: Laborczi Zoltán
okl. matematikus

© Seres Attila–Fenyő László–Gyalogh Kálmán
Budapest, 1986

ETO: 519.682 FORTH
800.92 FORTH

ISBN: 963 10 6858 7

Felelős szerkesztő: Votisky Zsuzsa

Tartalomjegyzék

1. A FORTH nyelv kialakulása..... 7
   1.1. A nyelv születése..... 7
   1.2. A FORTH mint önálló rendszer..... 9
   1.3. A FORTH-79 és egyéb változatok..... 15
      1.3.1. A FORTH-79 szabvány..... 15
      1.3.2. Egyéb változatok..... 15
2. A FORTH mint interpreter..... 17
   2.1. A verem fogalma és szerepe a FORTH-ban..... 17
      2.1.1. A fordított lengyel formátum fogalma. 19
      2.1.2. Kifejezések átirása fordított lengyel formára..... 20
   2.2. Adattipusok..... 23
      2.2.1. Standard adattipusok..... 25
         2.2.1.1. Egész tipus..... 25
         2.2.1.2. Duplapontos egész tipus... 26
      2.2.2. Egyéb adattipusok..... 28
         2.2.2.1. Valós tipus..... 28
         2.2.2.2. Szöveg tipus..... 29
   2.3. Változó- és konstansdefiniciók..... 30
      2.3.1. Memóriakezelés..... 31
         2.3.1.1. A memória irása és olvasása..... 31
         2.3.1.2. Egyéb memóriakezelő műveletek..... 33
      2.3.2. Konstansok..... 34
      2.3.3. Változók..... 35
   2.4. Veremkezelő műveletek..... 37
      2.4.1. Adatvermet kezelő műveletek..... 37
      2.4.2. Visszatérési vermet kezelő műveletek. 43
   2.5. Aritmetikai utasitások..... 44
      2.5.1. Egész tipusú műveletek..... 45
      2.5.2. Duplapontosságú aritmetika..... 49
      2.5.3. Vegyes hosszúságú aritmetika..... 50

Tartalomjegyzék

2.6. Relációk és logikai müveletek..... 52
2.6.1. Relációk..... 53
2.6.2. Logikai müveletek..... 54
2.7. Adatkonverziós müveletek..... 56
2.7.1. Konverzió egészből duplapontos egészbe..... 56
2.7.2. Konverzió számok és karakterfüzérek között..... 57
2.7.2.1. Számok átalakitása karakterfüzérré..... 57
2.7.2.2. Konverzió karakterfüzérből számba..... 61
2.7.3. Konverzió karakterből számjeggyé..... 62
2.8. Szövegek és karakterfüzér müveletek..... 63
2.9. Ki- és beviteli müveletek..... 65
2.9.1. Számok kiiratása..... 65
2.9.2. Karakterek és karakterfüzérek kivitele..... 67
2.9.3. Karakter- és karakterfüzér-bevitel... 68
2.10. Feladatok..... 69

3. A FORTH mint compiler..... 71
3.1. Algoritmusabsztrakció..... 71
3.1.1. A szó mint a FORTH program alapegysége..... 73
3.1.2. A szó mint eljárás..... 79
3.1.3. A veremszintek mint formális paraméterek..... 79
3.2. A szavak használata..... 81
3.2.1. A szótár felépitése..... 81
3.2.1.1. Általános ismertetés..... 81
3.2.1.2. Szótárkezelő utasitások... 83
3.2.2. Az újradefiniálhatóság és hatása..... 88

Tartalomjegyzék

3.3. A programvezérlés eszközei..... 95
3.3.1. Vezérlőutasitások..... 97
3.3.1.1. IF..ELSE..ENDIF, IF..ELSE..THEN..... 97
3.3.2. Ciklusszervező utasitások.....101
3.3.2.1. DO..LOOP, +LOOP, /LOOP....102
3.3.2.2. I, I', J, LEAVE.....104
3.3.2.3. BEGIN..WHILE..REPEAT.....108
3.3.2.4. BEGIN..UNTIL, BEGIN..END..109
3.3.2.5. BEGIN..AGAIN.....111
3.3.3. Rekurzió.....111
3.3.3.1. SMUDGE.....112
3.3.3.2. MYSELF.....114
3.4. Közvetlen szavak.....114
3.5. Forditásvezérlő utasitások.....118
3.5.1. COMPILE.....118
3.5.2. [COMPILE].....120
3.5.3. A [és a].....121
3.5.4. LIT, LITERAL, DLITERAL.....121
3.6. Rendszervezérlő utasitások.....126
3.6.1. ABORT, SP!, RP!.....126
3.6.2. QUIT.....127
3.6.3. COLD, WARM.....128
3.7. Feladatok.....129

4. A helyes programozási stilus FORTH-ban.....131
4.1. A "bottom-up" programozási módszer.....134
4.2. Moduláris programozás.....134
4.2.1. Logikailag különálló programegységek.134
4.2.2. A "top-down" programozási elv.....135
4.2.3. Alkalmazási példa.....136
4.3. A FORTH kiterjeszthetősége.....138

Tartalomjegyzék

5. Adatkezelés.....140
5.1. Helykijelölés.....140
5.1.1. ALLOT.....140
5.1.2. , és C,.....141
5.2. Adatszerkezetek létrehozása.....141
5.2.1. CREATE, <BUILDS..DOES>.....141
5.2.2. Példák adatszerkezetek létrehozására.144
5.2.2.1. n-elemű vektor létrehozása.....144
5.2.2.2. Karakterfüzér-konstans létrehozása.....145
5.3. Virtuális memóriakezelés.....146
5.3.1. Pufferek kezelése.....148
5.3.1.1. BLOCK.....148
5.3.1.2. BUFFER.....149
5.3.1.3. UPDATE.....149
5.3.1.4. SAVE-BUFFERS, FLUSH.....149
5.3.1.5. EMPTY-BUFFERS.....149
5.3.1.6. COPY.....149
5.3.1.7. WIPE.....150
5.3.1.8. R/W.....150
5.3.2. A screen fogalma.....150
5.3.3. Screenkezelés.....151
5.3.3.1. LOAD.....151
5.3.3.2. LIST.....151
5.3.3.3. INDEX.....153
5.3.3.4. TRIAD.....153
5.3.3.5. (LINE), .LINE.....153

5.4. A felhasználói szótár használata.....153
5.4.1. VOCABULARY.....156
5.4.2. DEFINITIONS.....156
5.4.3. Előre létrehozott szótárak a FORTH-ban.....158
5.4.3.1. A FORTH alapszótár.....158
5.4.3.2. A beépitett assembler.....158
5.4.3.3. A beépitett szövegszerkesztő.....164

6. A FORTH rendszer működése.....165
6.1. A FORTH kódhossza és futási ideje.....166
6.2. A "threaded code" fogalma.....170
6.3. A FORTH rendszer mint virtuális processzor....174
6.4. Az utasitás-elővételi algoritmus.....175
6.4.1. Az IP és W regiszterek szerepe. A NEXT ciklus.....178
6.4.2. A visszatérési verem szerepe. A DOCOL és a SEMIS.....181
6.4.3. DOVAR, DOCON, DOUSE.....186
6.5. A szó belső ábrázolása.....189
6.5.1. Az nfa, lfa, cfa, pfa fogalma és használata.....189
6.5.2. Utasitások a szó belső elemeinek elérésére.....195
6.6. Vezérlésátadás a felfűzött kódban.....200
6.6.1. Ugrási eltolás számitása a BACK szóval.....203
6.6.2. EXECUTE.....208
6.6.3. INTERPRET.....210
6.7. Rendszerváltozók és rendszerkonstansok.....212
6.7.1. Rendszerváltozók.....212
6.7.2. Rendszerkonstansok.....217
6.8. Egyéb szavak.....218

Tartalomjegyzék

Függelék:

F1. Feladatok megoldásai.....222
F2. Példaprogram.....229
F3. Konkrét FORTH implementációk
F3.1. 8080 CPM FORTH 2.2.....233
F3.2. Commodore C-64 FORTH.....239
F3.3. Spectrum ABERSOFT FORTH.....246
F3.4. ZX SPECTRUM Floating Point FORTH.....255
F3.5. SIF-Z80 FORTH 2.2.....268
F4. A FORTH szavak betűrendben.....279
F5. A FORTH-79 fig-FORTH-tól eltérő utasitásainak definiciója.....289
F6. A FORTH rendszer memóriaképe.....290
F7. Irodalomjegyzék.....291

1. A FORTH nyelv kialakulása

1.1. A nyelv születése

A FORTH nyelv szülőatyja, Charles Henry Moore egyesült államokbeli csillagász. Ez a tudós végtelenül elégedetlen volt korának legelterjedtebb programozási nyelveivel. Ezek a nyelvek (pl. FORTRAN, ALGOL, COBOL) nem biztositották azt a könnyedséget, rugalmasságot és ugyanakkor nagy sebességet, ami a hatékony programozói munkához elengedhetetlenül szükséges. Talán a legfontosabb szempont, hogy a nyelv utasitáskészlete korlátlanul kiterjeszthető legyen adott feladatok speciális igényeinek megfelelő módon.

Miután tudatosultak benne ezek a problémák, hozzáfogott egy olyan programozási nyelv kifejlesztéséhez, amely a fenti elvárásoknak a lehető legjobban megfelel. Munkáját az IBM 1130 (3. generációs) számitógépen végezte. A program-rendszer 1968-ban készült el. Moore úgy érezte, hogy olyan nyelvet alkotott, amely a "majdani" 4. generációs számitógépek általánosan használt programozási nyelve lehet. Ezért a nyelvet FOURTH-nak (= negyedik) szerette volna elnevezni, azonban az IBM 1130 operációs rendszere (job control) csak legfeljebb ötkarakteres azonositókat fogadott el. Igy lett a FOURTH-ból **FORTH**, amely nem kevésbé kifejező, jelentése: tovább, előre.

Kezdetben nagy lelkesedés övezte a nyelvet, azonban ez rövid idő alatt lelohadt. Az ok abban kereshető, hogy a nyelv legfőbb előnyei, úgymint a tömörség és a nagy sebesség nem bizonyultak fontosabbnak, mint a korabeli nyelvek könnyebb elsajátithatósága a nagyszámitógépek korában. Ezenkivül ebben az időben kevés helyen álltak a programirók rendelkezésére olyan interaktiv eszközök (képernyő, billentyűzet), amelyek a FORTH használatához elengedhetetlenül szükségesek.

1.1. A nyelv születése

A FORTH a mikroszámitógépek megjelenésével került ismét az érdeklődés középpontjába az 1970-es évek végén. Itt már a szűkös memória, és a viszonylag lassú központi egység jobb kihasználása tömör és gyors programozási nyelvet igényelt. Ezeknek a követelményeknek az addig méltatlanul mellőzött FORTH nyelv minden szempontból megfelel. Igy lett - a BASIC mellett - a FORTH a személyi számitógépek legáltalánosabban használt nyelve.

Végezetül álljon itt egy idézet az alkotótól a Starting FORTH cimű könyv előszavából:

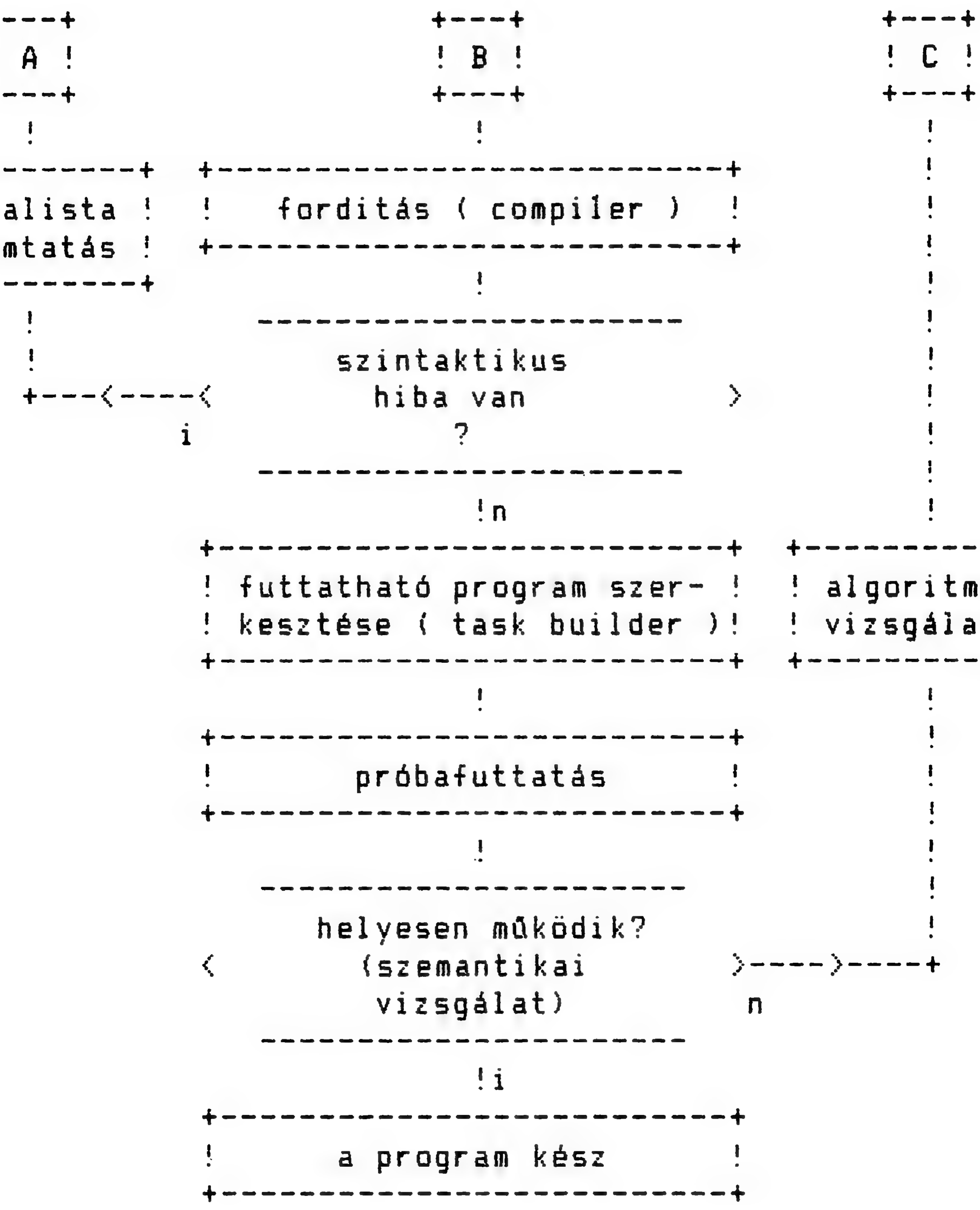
> "Bár én vagyok az egyetlen ember, akinek sohasem kellett megtanulnia a FORTH-t, mégis tudom, hogy elsajátitása nem egyszerű dolog. Ugyanúgy, mint az emberi nyelvben, itt is sok szó jelentését kell megjegyezni. [...] A FORTH biztositja a természetes kapcsolatot az ember és az őt körülvevő eszes gépek között. Ez megkivánja az emberi nyelv olyan tulajdonságait, mint a tömörség, a rugalmasság és a kiterjeszthetőség. Nem tudok elképzelni jobb nyelvet a programiráshoz, algoritmusok kifejtéséhez és a számitógépek működésének megértéséhez."

1.2. A FORTH mint önálló rendszer

Az alábbiakban hasonlitsuk össze a FORTH rendszer programfejlesztési struktúráját más, elterjedt rendszerekével.

Forditóprogramok:

A magasszintű struktúrált nyelvek forditóprogramjai (compiler) általában az 1.1. árán látható programfejlesztési struktúrát követelik meg.



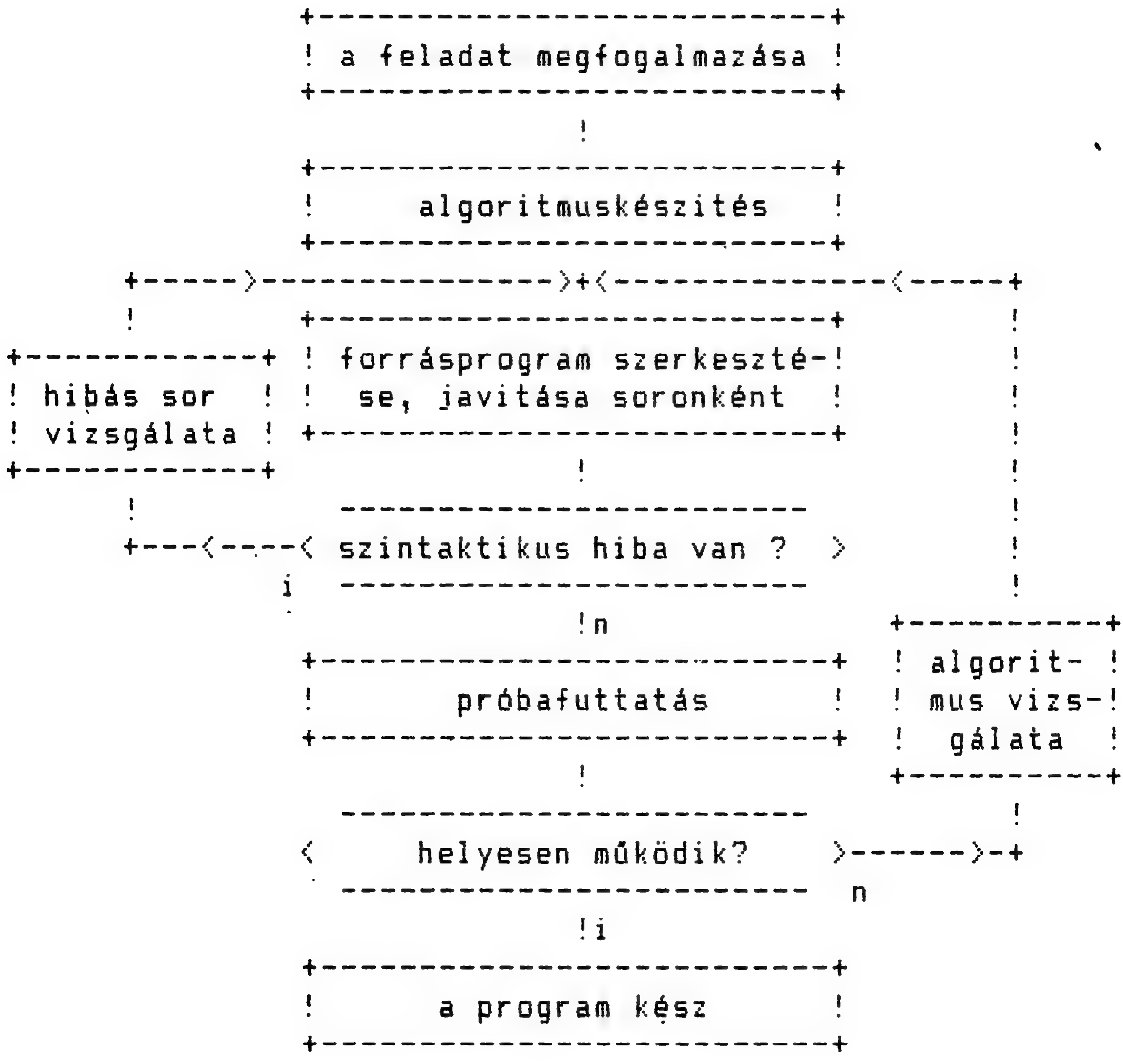
1.1. ábra
Forditóprogramok programfejlesztési struktúrája

1.2. A FORTH mint önálló rendszer

A feladat megfogalmazása után ki kell dolgozni egy megfelelő algoritmust a megoldás menetére. Ezt követően behivjuk a szövegszerkesztőt a memóriába, és elvégezzük a forrásprogram szerkesztését. Kilépünk az editorból, majd a fordítóprogram segitségével előállitjuk a program tárgykódját és a hibalistát. A fordítás során a compiler jelzi, ha szintaktikus hibát talált. Ez esetben kinyomtatjuk a hibalistát, és általában hosszasan merengünk a hibajelzések okán, majd újra betöltjük a szövegszerkesztőt, javitunk, forditunk és igy tovább... Ha végre sikerült hiba nélkül leforditani a forrásprogramot, a szerkesztő programmal a tárgykódból megszerkesztjük a futtatható programot. Végre elinditjuk a próbafutást, majd az esetek nagy részében szomorúan tapasztaljuk, hogy az nem az elképzeléseinknek megfelelően működik. Ekkor megpróbáljuk hideg fejjel újra átgondolni az algoritmust. Amikor úgy véljük, hogy megtaláltuk a hibát, betöltjük a szövegszerkesztőt, javitunk, forditunk és igy tovább... amig hosszas fejlesztés után végre megszületik a kivánt program.

Értelmező programok:

Értelmező program (interpreter) esetén a 1.2. ábra szerint módosul az eljárás.



1.2. ábra
Értelmező programok programfejlesztési struktúrája

A fejlesztés első két lépése megegyezik a fordítóprogramoknál leirtakkal. Az interpreterek esetén általában nincs szükség külön szövegszerkesztő behivására, mivel az állandóan a memóriában van. Igy soronként szerkeszthetjük a forrásprogramot, és a szintaktikus hibákról többnyire azonnal jelzést kapunk, azt azonnal kijavithatjuk. Miután elkészültünk, egy egyszerű parancs segitségével azonnal futtathatjuk a programot. Ha szemantikai hibát találunk, önkritikusan meg kell vizsgálnunk az elképzelt algoritmust, majd korrigálnunk a hibákat.

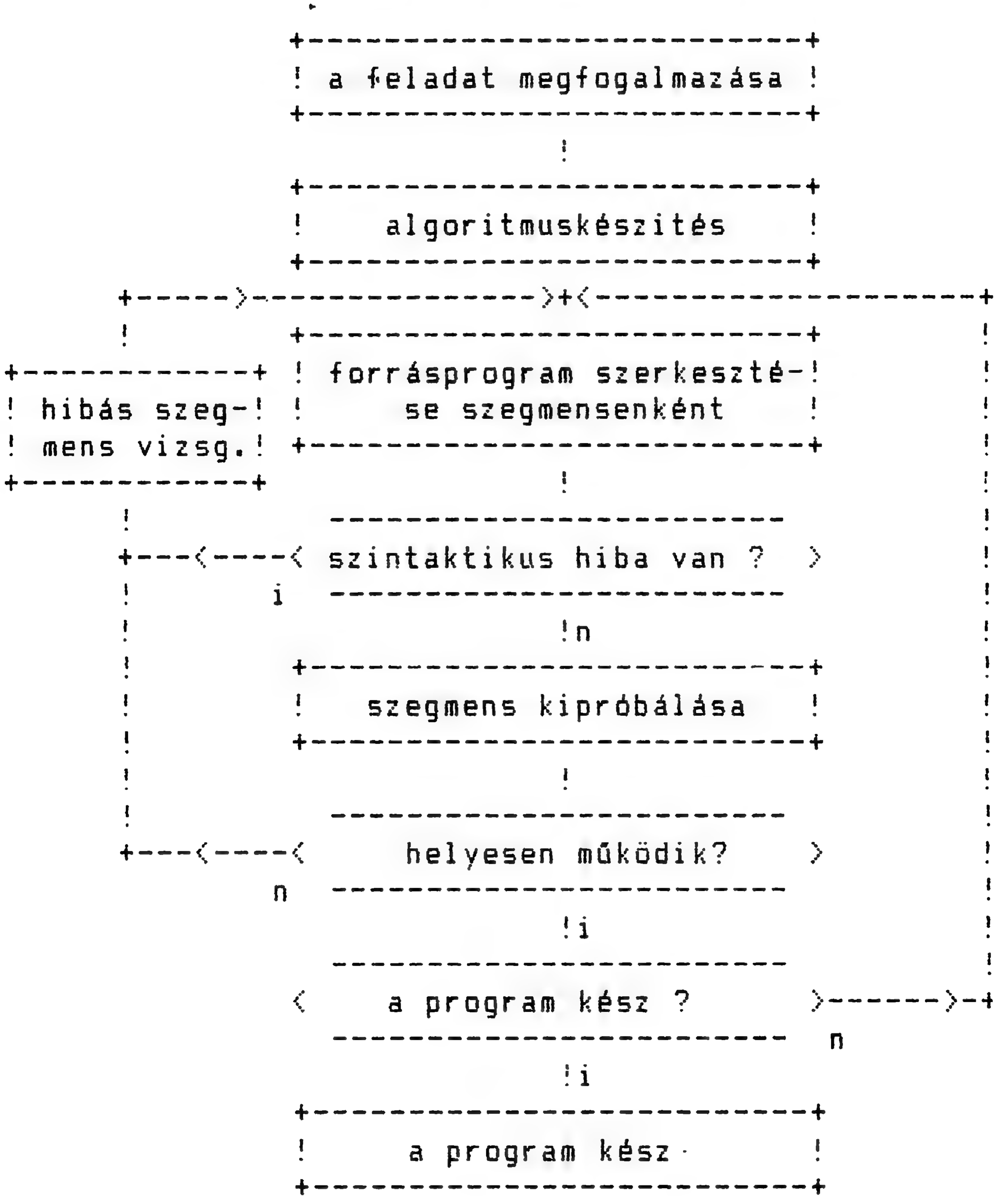
Látható, hogy a compilerek sokkal nehézkesebb, hosszadalmasabb programfejlesztést kivánnak, mint az interpreterek. Utóbbiak a felhasználóval interaktivan kommunikálnak, igy a forrásprogram folyamatosan javitható. Ennek az előnynek azonban igen nagy ára van! Az igy fejlesztett programok többszörösen lassabbak a compilerek által leforditott változatoknál. Ezenkivül a forrásprogram futás közben is a memóriában van, és sokkal több helyet foglal el, mint a tárgykód. Végül pedig az interpreter is jelentős memóriaterületet foglal el.

A FORTH rendszer

A FORTH igen szerencsésen ötvözi a kétféle program-fejlesztési struktúra előnyeit, hátrányait pedig szinte teljesen kiküszöböli (1.3. ábra).

A felhasználó számára a FORTH interpreterként viselkedik, azonban az egyes eljárások (szegmensek) szerkesztése automatikusan forditás (compiler) üzemmódban történik.

1.2. A FORTH mint önálló rendszer



1.3. ábra
A FORTH rendszer programfejlesztési struktúrája

1.3. A FORTH-79 és egyéb változatok

1.3.1. A FORTH-79 szabvány

A FORTH rugalmassága és kiterjeszthetősége azt a következményt vonta maga után, hogy gomba módra szaporodtak a különböző, egymással egyre kevésbé kompatibilis FORTH rendszerek. A FORTH-ban megirt programok nem voltak hordozhatóak. Ez a helyzet egyre tarthatatlanabbá vált, igy rövidesen felmerült a szabványositás igénye.

Ezzel először 1973-ban próbálkoztak az egyesült államokbeli Arizona államban levő Kitt Peak Observatory (Ch. Moore munkahelye) munkatársai. Kisérletük sikertelen volt. Legközelebb csak 1978-ban született újabb javaslat, amelyet az Európai és Amerikai FORTH Felhasználók Szervezete (European and American FORTH Users) dolgozott ki, és jelentetett meg FORTH-77 szabvány elnevezéssel. Ennek tovább-fejlesztett változata volt a FORTH-78, amelyet a FORTH Nemzetközi Szabványositási Csoport (FORTH International Standards Team) hozott létre.

Ezen előzmények után fejlesztette ki 1979-ben a ma is általánosan elfogadott FORTH szabványt a FORTH Szabványositási Csoport (FORTH Standards Team). A szabvány neve: FORTH-79 STANDARD. A szabvány kötelező előirásokat valamint ajánlásokat tartalmaz. Könyvünk mind a kötelező, mind az ajánlott szavak, valamint a legelterjedtebb rendszer, a fig-FORTH ezektől eltérő utasitásainak leirását tartalmazza.

1.3.2. Egyéb változatok

Fel kell hivnunk az Olvasó figyelmét, hogy a könyvünkben leirt szavak némelyike feltehetően nem található

meg saját FORTH rendszerében, ill. nem az itt megadott módon működik. Ennek magyarázata az, hogy a szabványos FORTH rendszerek sem mindig teszik magukévá a FORTH-79 szabvány ajánlásait. és akkor még nem is beszéltünk a nem szabványos FORTH rendszerekről.

A fenti problémákra való tekintettel az F3. függelékben leirjuk a Magyarországon leginkább elterjedt FORTH rendszerek sajátosságait. Ezen túlmenően tárgymutatóként szolgálhat az F4. függelék betűrendes utasitás összefoglalója. Itt felsoroljuk a részletesen vizsgált szavakat megjelölve, hogy azok amely FORTH rendszerekben szerepelnek. A fig-FORTH tulajdonosok kedvéért az F5. függelékben leirtuk a fig-FORTH-ban nem szereplő FORTH-79 szavak definicióit. Az F6. függelék a FORTH rendszer memóriaképét mutatja be. Azok számára, akiknek érdeklődése túlmutat e könyvön, az F7. irodalomjegyzékben felsoroltunk néhány, könyvünk alapjául szolgáló művet. A 2. és 3. fejezet feladatainak megoldása az F1. függelékben található. Végül megemlitjük, hogy az F2. függelékben bemutatunk egy hasznos FORTH programot.

2. A FORTH mint interpreter

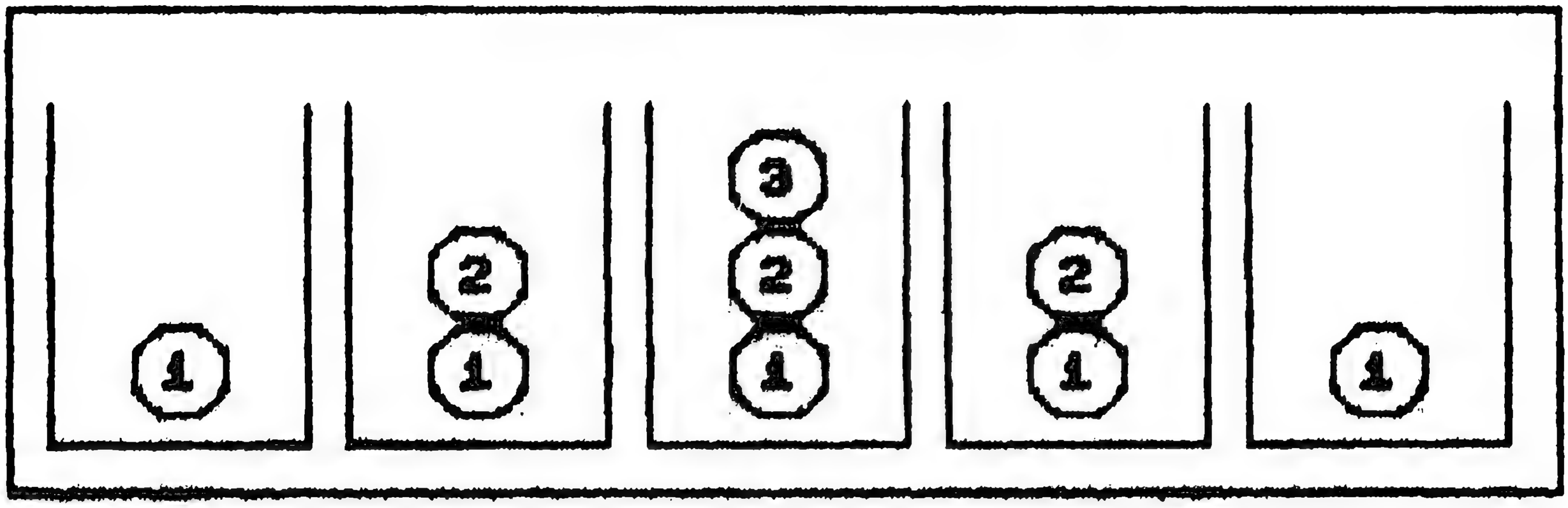
A FORTH interpreter (= értelmező) üzemmódban a következőképpen használható: egyszerűen leirunk számokat, ill. utasitásokat egymástól szóközzel elválasztva, majd a RETURN gomb (= ENTER, CR stb.) lenyomásakor az interpreter azonnal végrehajtja az előirt műveletsort. Ahhoz, hogy ezt a gyakorlatban is kipróbáljuk, meg kell ismerkednünk néhány alapfogalommal.

2.1. A verem fogalma és szerepe a FORTH-ban

A számitástechnikában használatos verem leginkább az alábbi példával illusztrálható:

Képzeljünk el egy teniszlabdák tárolására szolgáló dobozt (2.1. ábra). A labdákat egymás után helyezhetjük el a dobozban, igy először az utolsóként berakott labdát vehetjük ki. Ezt a tárolási elvet a szakirodalomban LIFO-nak nevezik (Last In First Out = utolsóként be, elsőként ki). Ha egy mélyebben elhelyezett labdát szeretnénk elővenni, ehhez az összes fölötte levőt ki kell emelnünk a dobozból.

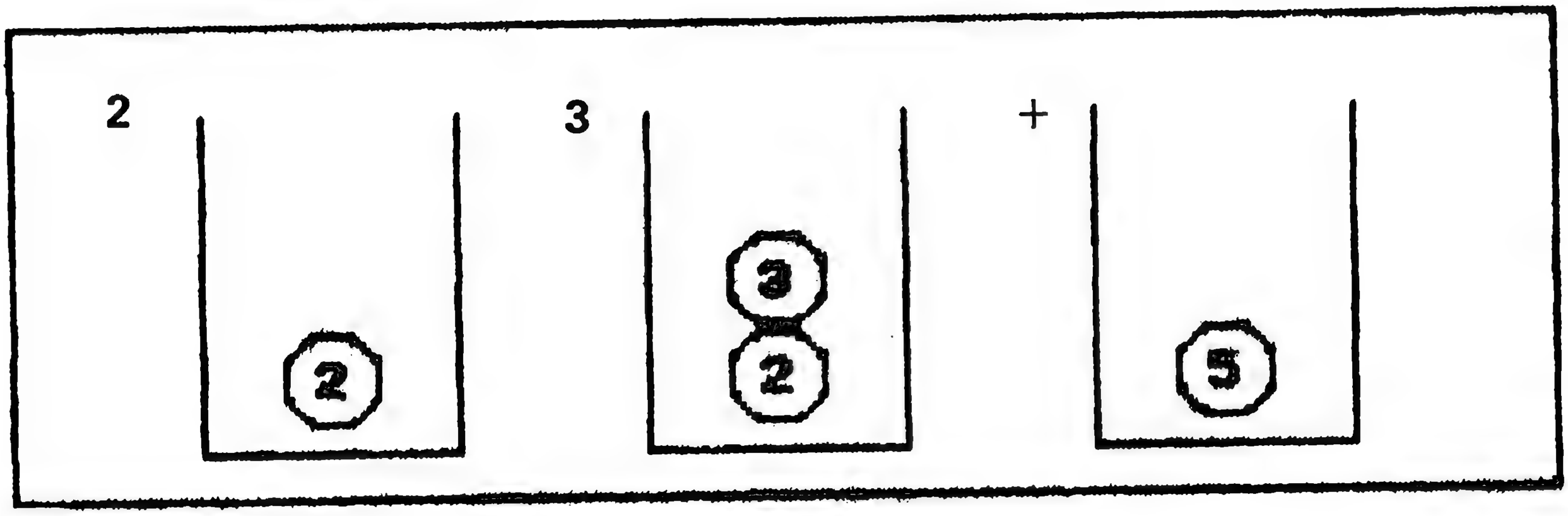
2.1. A verem fogalma és szerepe a FORTH-ban



2.1. ábra
Adatok tárolása a veremben

A valóságban a fenti példában tárgyalt doboz egy memóriaterületnek, a teniszlabdák pedig számoknak felelnek meg. Ez a FORTH-ban is igy van. A nyelv két független vermet használ. Az egyik a FORTH rendszer működéséhez szükséges. Ezt visszatérési veremnek (return stack) nevezzük, és szerepét részletesen a 6.4.2. pontban ismertetjük. A másik az adatverem (data stack), amit kitüntetett szerepe miatt a továbbiakban egyszerűen csak veremnek fogunk nevezni.

A FORTH nyelvben műveleteket csak a veremben levő számokkal végezhetünk. Például, ha végre akarjuk hajtani a 2+3 műveletet, ehhez először a 2 és a 3 számokat a verem tetejére kell helyeznünk, majd kiadhatjuk az összeadás elvégzésére a + utasitást. Ez a művelet elveszi a verem tetejéről a két számot, majd ezek helyére elhelyezi az összeadás eredményét, példánkban az 5-öt (2.2. ábra).



2.2. ábra
Veremállapotok egy összeadás végrehajtása során

2.1.1. A forditott lengyel formátum fogalma

Az 1950-es években az elméleti matematikusok kidolgozták a vermen alapuló, úgynevezett veremgépek elméletét. Közöttük volt J. Lukasiewicz lengyel származású matematikus, aki kidolgozta a veremgépekhez illeszkedő aritmetikai jelölésformát, amely forditott lengyel formátum néven vonult be a szakirodalomba (reverse Polish notation).

Térjünk vissza az előző pontban tárgyalt példára! A megszokott módon igy irnánk fel a műveletet: 2 + 3. A FORTH rendszerben mint láttuk, a művelet elvégzése előtt el kell helyeznünk az operandusokat a veremben. Ezért esetünkben - bár ez első pillantásra talán meglepőnek tűnhet - igy alakul a formátum: 2 3 + .

A FORTH-ban az összes aritmetikai műveletre igaz az, hogy előbb le kell irnunk a megfelelő számú operandust, és azután jelölhetjük ki a műveletet. Belátható, hogy

tetszőlegesen bonyolult, bármilyen mélységben zárójelezett aritmetikai kifejezést átirhatunk forditott lengyel formátumra.

2.1.2. Kifejezések átirása forditott lengyel formára

Először tekintsük át a precedencia fogalmát. Ez a különböző aritmetikai és logikai műveletek végrehajtási sorrendjét definiálja. A számitástechnikában kialakult precedenciaszabályok a következők:

a. mindig a magasabb precedenciájú műveletet kell előbb elvégezni.
b. az azonos precedenciájú műveleteket balról jobbra haladva kell elvégezni
c. a zárójeles kifejezéseket a legbelsőtől a legkülső zárójel felé haladva kell kiértékelni az a. és b. pontok figyelembevételével

^	hatványozás
-	ellentett képzés (pl. -3)
* /	szorzás, osztás
+ -	összeadás, kivonás
NOT	logikai NEM
AND	logikai ÉS
OR	logikai VAGY

Most tekintsünk egy bonyolultabb aritmetikai kifejezést:

(6 + 4 * (36 / 2 - 7) - 4) / 2

A kiértékelés az alábbi lépésekben történik:

```
( 6 + 4 * (      18 - 7 ) - 4 ) / 2
( 6 + 4 * (      11     ) - 4 ) / 2
( 6 +             44       - 4 ) / 2
(                 50       - 4 ) / 2
(                 46         ) / 2
                  23
```

Ezekután a forditott lengyel formára való átirás szabálya a következőképpen alakul.

A kifejezéskiértékelés lépéseit a fenti sorrendben végrehajtjuk, a műveletek végrehajtása helyett azonban forditott lengyel formára alakitjuk a szóban forgó részkifejezést oly módon, hogy a korábban már átalakitott részeket változatlanul hagyjuk:

(6 + 4 * (36 / 2 - 7) - 4) / 2

Az átirás a következő lépésekben történik:

```
( 6 + 4 * ( 36 2 / - 7 ) - 4 ) / 2
( 6 + 4 * ( 36 2 / 7 - ) - 4 ) / 2
( 6 +   36 2 / 7 - 4 *   - 4 ) / 2
(   36 2 / 7 - 4 * 6 +   - 4 ) / 2
(   36 2 / 7 - 4 * 6 + 4 -   ) / 2
36 2 / 7 - 4 * 6 + 4 - 2 /
```

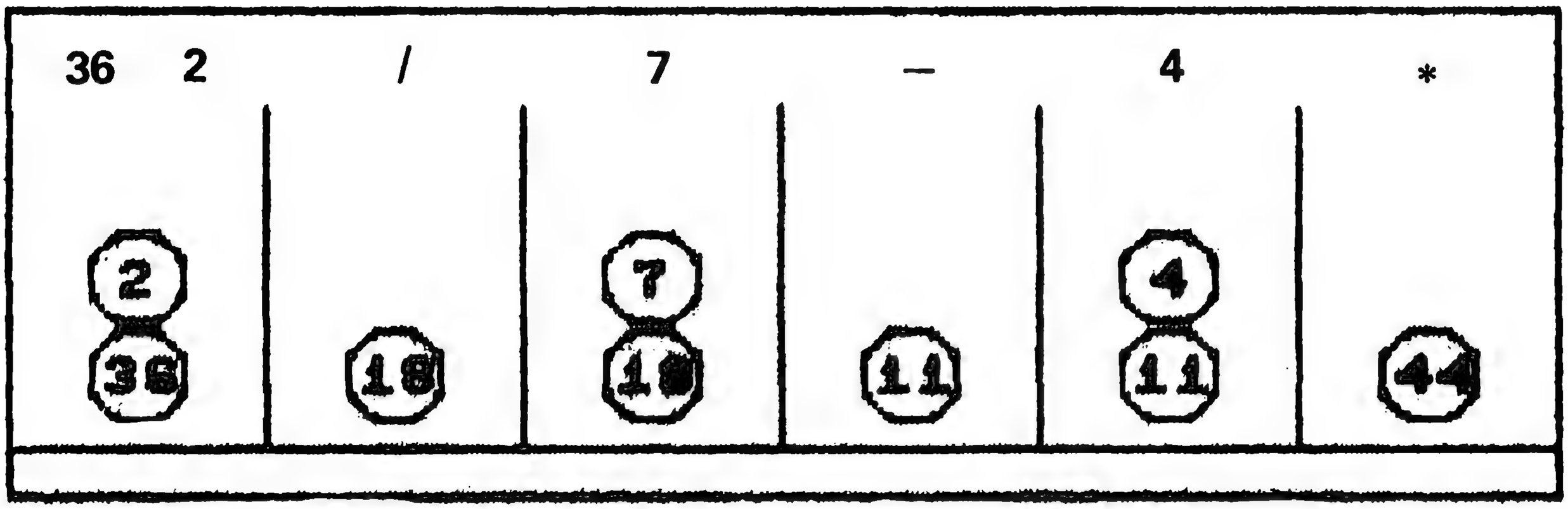
Ezzel elő is állt az első, az interpreter számára is érthető FORTH utasitássorozatunk. Ha ezt leirjuk, a verem tetején megjelenik a 23 szám. Hogyan tudjuk ezt kiiratni a képernyőre? Erre a választ részletesen a 2.9. szakaszban

adjuk meg. Elöljáróban azonban ismerkedjünk meg a verem tetején levő szám kiiratására szolgáló FORTH utasitással, amely a pont (.)!

Most összefoglalásképpen nézzük meg az utasitássorozat hatását a FORTH interpreter felügyelete alatt. A futási listákon a továbbiakban aláhúzással jelöljük a FORTH rendszer által kiirt információkat. A felhasználónak előtte le kell ütnie a RETURN billentyűt.

36 2 / 7 - 4 * 6 + 4 - 2 / . 23 ok

A 2.3. ábrán nyomonkövethetjük a verem állapotait az utasitássorozat végrehajtása során. (A veremben mindig a fölötte megadott utasitás végrehajtása utáni állapotot láthatjuk.)



2.3. ábra
Veremállapotok egy utasitássorozat végrehajtása során

2.2. Adattipusok

Programirás során többféle adat tárolására, kezelésére van szükség. A magasszintű programnyelvek általában az alábbi adattipusokat különböztetik meg:

a) egész (integer),

b) logikai (Boolean),
c) valós (real),
d) szöveg,karakterfüzér (string,text,char).

A FORTH elsősorban az egész tipusú adatok feldolgozását támogatja. Ez a gépi kóddal való rokonságával magyarázható, ugyanis a számitógépek központi egysége többnyire csak ezt a tipust ismeri. A valóságban ez a korlát nem jelent túl nagy megkötést, mivel a többi adattipus is ábrázolható egész tipusú számokkal. Nézzük sorra:

b) logikai adat, értéke igaz (true), vagy hamis (false). Az igaz értékekhez egyértelműen hozzárendelhető a nullától különböző egész számok halmaza, ill. a hamishoz a nulla (0);

c) valós tipus: a FORTH megvalósitások általában kétféleképpen oldják meg ennek a tipusnak a használatát. Legtöbb esetben a duplapontos egész tipussal (double-length integer) helyettesitik, ritkábban lebegőpontos számábrázolást biztositanak;

d) szöveg tipusú adatokat az alábbi módon lehet tárolni: a szöveg egyes elemeit (betűket, karaktereket) egész számokkal (ASCII kóddal) helyettesitjük, és egy további egész számban megjegyezzük a szöveg hosszát.

Az előbbieknek megfelelően vizsgáljuk meg részletesen a FORTH adatok alaptipusait (standard adattipusok), majd az ezektől eltérő egyéb tipusokat, külön kifejtve ezek ábrázolását a memóriában. Itt szükséges megemliteni, hogy ennek a memóriaképnek az ismerete nem feltétlenül szükséges a FORTH programozáshoz.

2.2.1. Standard adattipusok

2.2.1.1. Egész tipus

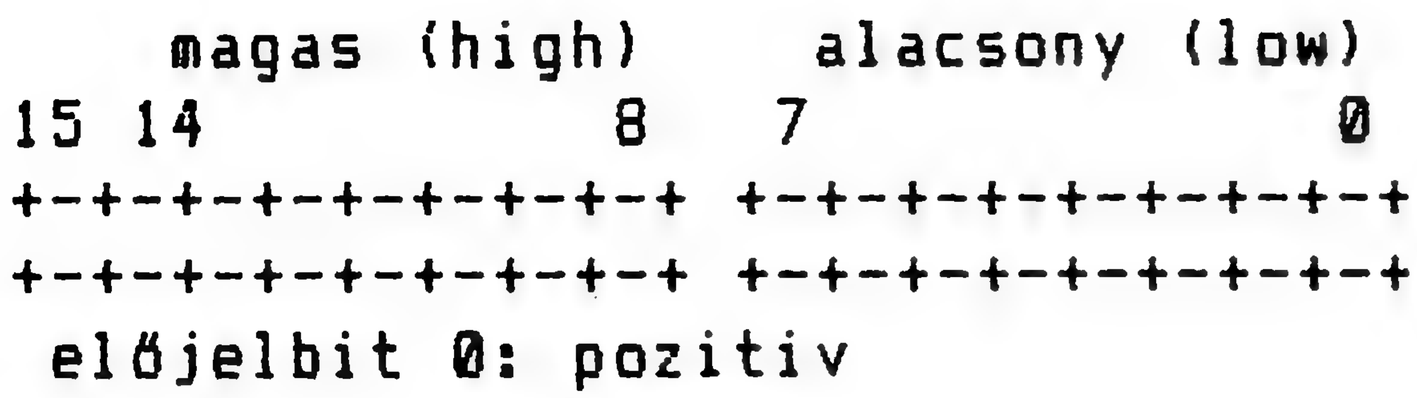
Az FORTH a beállitott számrendszernek megfelelő formátumú számokat egész számként értelmezi. (A számrendszer beállitását a 2.6. szakaszban részletezzük.) Példák:

```
1985     ( tizes számrendszer )
3D4A     ( tizenhatos számrendszer )
10010001 ( kettes számrendszer )
```

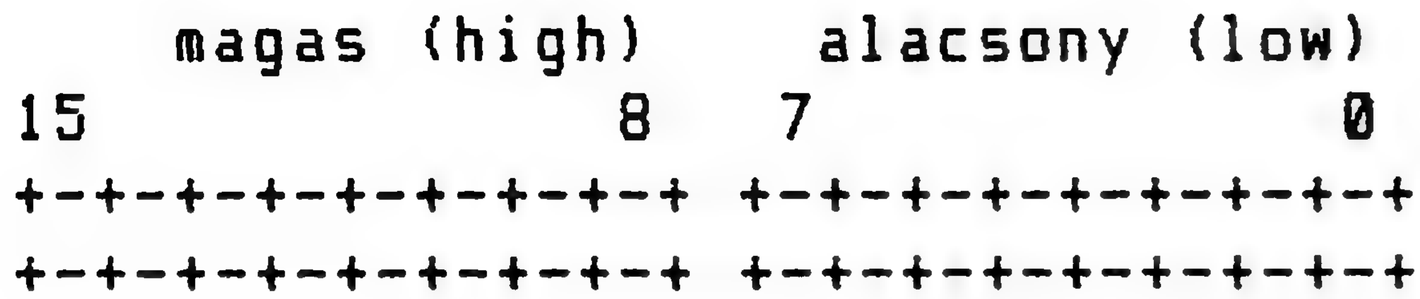
A FORTH az egész tipusú számokat két bájton, azaz 16 biten tárolja. Ezt kétféleképpen lehet értelmezni:

a) Előjeles egész
A FORTH-ban az ábrázolható legkisebb és legnagyobb előjeles egész szám -32768, ill. +32767. A szám tárolása a memóriában a 2.4a ábrának megfelelően történik. A 0...14. biteken helyezkedik el a pozitiv szám abszolút értéke, ill. a negativ szám abszolút értékének 2-es komplemense, a 15. bit az előjel.

b) Előjel nélküli egész
Az előjel nélküli egész számok értékei a 0...65535 számtartományba esnek. Ábrázolásukat a 2.4b ábrán láthatjuk.



2.4a ábra
Előjeles egész számok ábrázolása



2.4b ábra
Előjel nélküli egész számok ábrázolása

2.2.1.2. Duplapontos egész tipus

A duplapontos egész számok formátumát az különbözteti meg a szimpla egész számoktól, hogy a számjegyek között - tetszőleges helyen - egy pont szerepel. Példák:

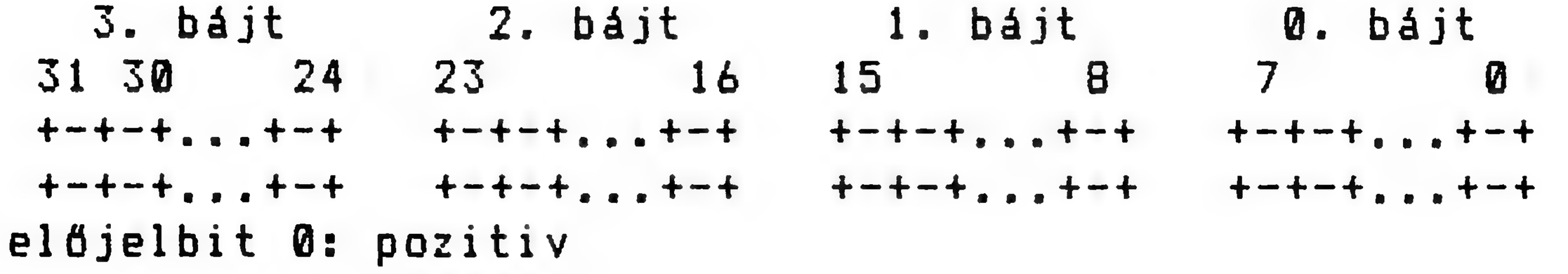
```
198.5     ( tizes számrendszer )
1985.     ( = 198.5 = 19.85 = 1.985 !!! )
3D4.A     ( tizenhatos számrendszer )
3D4A.     ( = 3D4.A = 3D.4A = 3.D4A !!! )
100.0001  ( kettes számrendszer )
```

A FORTH a duplapontos egész számokat 32 biten, azaz 4 bájton tárolja. A 16 bites egészhez hasonlóan ez is kétféle-

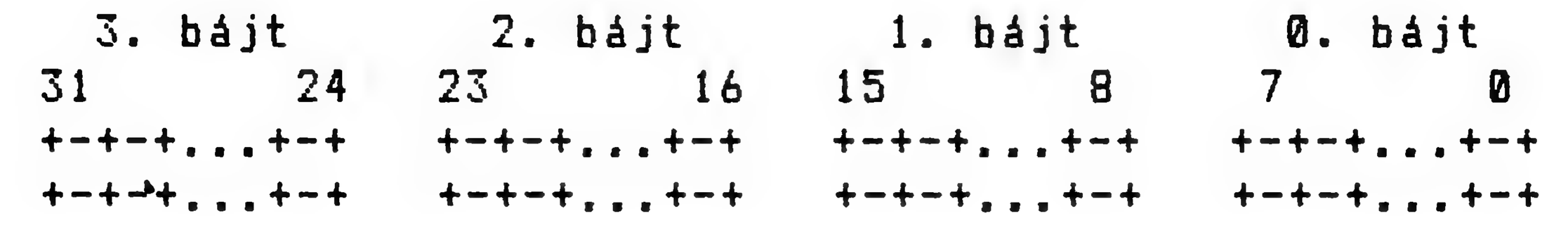
képpen értelmezhető:

a) Előjeles duplapontos egész
Ez a tipus a $-2^{31}...+2^{31}-1$, azaz a -2 147 483 648... +2 147 483 647 számtartományt foglalja magába. A 0...30. biteken helyezkedik el a pozitiv duplapontos egész szám abszolút értéke, ill. a negativ szám abszolút értékének 2-es komplemense, a 31. bit az előjel. Ábrázolását a 2.5a ábrán figyelhetjük meg.

b) Előjel nélküli duplapontos egész
Az előjel nélküli duplapontos egész számok értékei a $0...2^{32}-1$ (0... 4 294 967 295) számtartományba esnek. Ábrázolásukat a 2.5b ábrán láthatjuk.



2.5a ábra
Duplapontos előjeles egész számok tárolása



2.5b ábra
Duplapontos előjel nélküli egész számok tárolása

Meg kell jegyeznünk, hogy bizonyos FORTH megvalósitások nem ismerik a duplapontos egész adattipust. Ez általában a lebegőpontos (floating point) FORTH rendszerekben fordul elő, ahol e tipust a valós tipus helyettesiti.

2.2.2. Egyéb adattipusok

2.2.2.1. Valós tipus

A szabványos FORTH rendszerek általában nem ismerik ezt a tipust, használatát azonban a duplapontos egész adattipussal támogatják. Igy a valós tipust csak a függelékben szereplő lebegőpontos FORTH rendszereknél ismertetjük. Itt csak a szabványos FORTH által biztositott lehetőséget tárgyaljuk.

A programozó a duplapontos egész tipus segitségével könnyen használhat valós számokat is, azokat normált alakjukkal helyettesitve. Például:

$$3.14159265 = 314159265 * 10^{-8}$$

duplapontos egész — kitevő-rész

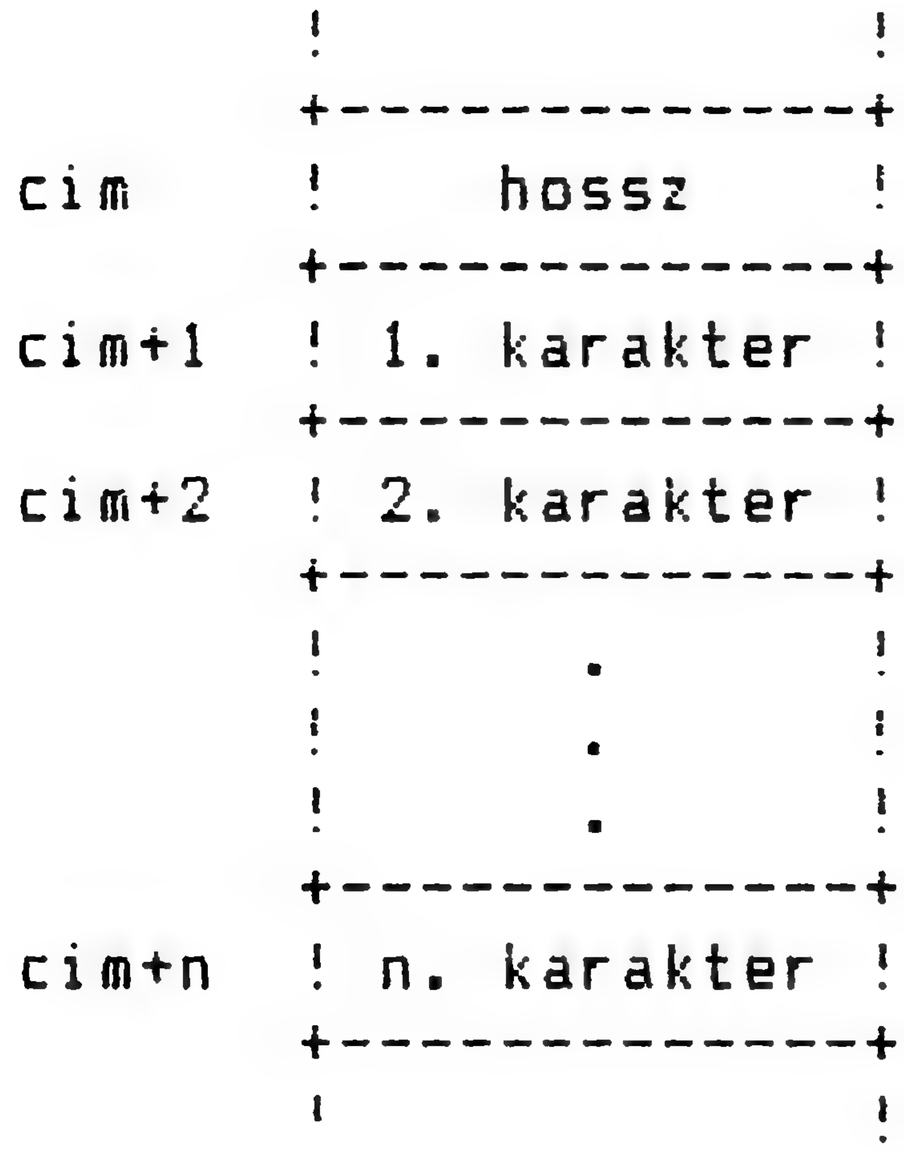
Természetesen a kitevőrész kitevőjét külön egész számként kell értelmezni. Ennek gyakorlati megvalósitása a későbbi fejezetek során tisztázódni fog.

Mi a magyarázata a lebegőpontos számábrázolás háttérbe szorulásának a FORTH-ban? A FORTH megalkotója abból indult ki, hogy ez a nyelv minél közelebb álljon a gép saját nyelvéhez. Nyilvánvaló, hogy a csak fixpontos számokkal dolgozó gép számára az állandó adatkonverzió a lebegőpontos számoknál komoly időveszteséget jelent. A gyakorlat igazolta Ch. Moore elgondolását. A gyakorlott FORTH programozók számára a fixpontos aritmetika használata ugyanolyan

természetes, mint másoknak a lebegőpontos. Ezt bizonyitja, hogy FORTH nyelven már olyan komoly matematikai, fizikai problémákat is megoldottak, mint a gyors Fourier transzformáció (FFT), lineáris regresszió, stb. A sebességnövekedés jól mérhető: a 32 bites fixpontos összeadás 5-10-szer gyorsabb a lebegőpontos változatnál.

2.2.2.2. Szöveg típus

A FORTH-ban a szöveg tipusú adatok tárolása karakterfüzér formájában történik. Más programozási nyelvekkel ellentétben a FORTH nem ismer előre definiált szöveg tipusú adategységet (változó, konstans). A karakterfüzér tárolása a memóriában a 2.6. ábrán látható módon történik.



2.6. ábra
Karakterfüzér belső ábrázolása

2.2. Adattipusok

Mivel a karakterfüzér hosszát 1 bájton ábrázoljuk, legfeljebb 255 karaktert helyezhetünk el egy füzérben. A FORTH a szöveg tipusú adatok számára egy memóriarészt tartott fenn (szövegpuffer, l. Fő. függelék). Ennek kezdőcimét a **PAD** utasitás (--- cim) adja meg. Jegyezzük meg, hogy értéke mindig változik. (Ennek az a magyarázata, hogy a szövegpuffer a későbbiekben tárgyalt felhasználói szótár végétől fix távolságra helyezkedik el, és ahogy ez a szótár növekszik, úgy "tolja" maga fölött a PAD-et.)

2.3. Változó és konstansdefiniciók

Ebben a fejezetben megkezdjük a FORTH rendszer utasitásainak ismertetését. A jobb áttekinthetőség érdekében a verem állapotainak ábrázolására egységes jelölést fogunk használni:

x1 x2 ... xn --- y1 y2 ... yn

ahol a --- jeltől balra levő x1 x2 ... xn a veremnek az utasitás végrehajtása előtti, a jobbra eső y1 y2 ... yn pedig a végrehajtás utáni logikai szintjeit jelöli. Balról jobbra haladva helyezkednek el a magasabb szintek, igy a verem tetején az xn ill. yn adatok találhatók. Az egész számokat n n1 n2..., a duplapontos egészeket d d1 d2... jelöli. Speciális jelentésű adatok esetén magyar kifejezést használunk (pl. egy memóriacimet a cim szóval jelölünk).

2.3.1. Memóriakezelés

A FORTH rendszert leggyakrabban 8-bites mikroszámitógépeken valósitják meg, igy a memóriakezelés is ehhez a struktúrához igazodik. A memória egyes rekeszeit 16 bites cimen érhetjük el (0...65535), az egyes rekeszek pedig 8 bites adatot tartalmaznak (1 bájt, értéke 0...255 lehet). A FORTH általában egy-egy rekeszpárt használ egyszerre, ez megfelel a szimpla hosszú egész szám formátumának (l. a 2.2.1.1. pontot).

2.3.1.1. A memória irása és olvasása

A memóriából kiolvashatunk, ill. oda beirhatunk adatokat. Ezeket a műveleteket az alábbi FORTH szavakkal végezhetjük el:

Memóriakiolvasás: **@** (cim --- n)
Az utasitás bemenő paramétere a vermen levő szám, amely egy memóriacimet reprezentál, kimenő paramétere a cimen és a következő cimen található két bájtnyi 16 bites adat, amelyet egész számként a veremre helyez.
A fenti utasitás 8-bites változata a **C@** (cim --- n). Ez a verem tetején levő cimen található egy bájtnyi adatot kiolvassa a memóriából, és ezt egész számmá kiegészitve a verem tetejére helyezi.
Dupla hosszú egész számok memóriából való kiolvasására szolgál a **2@** (cim --- d). Ez az utasitás a verem felső szintjéről vett cimtől kezdődő négy bájtnyi adatot duplapontos egész számként hozza fel a veremre.

Memóriabeirás: **!** (n cim ---)
Az utasitás bemenő paraméterei sorrendben: adat cim. A verem tetején levő és az azt követő cimre leteszi a 16

bites adatot alacsony bájt, magas bájt sorrendben.
Ennek 8 bites megfelelője a C! (n cim ---), amely a cimre helyezi az adat alsó 8 bitjét.
A 2! (d cim ---) utasitás a verem második és harmadik szintjét elfoglaló duplapontos egész számot a verem felső szintjén található cimtől kezdve helyezi a memóriába.

Speciális memóriabeiró utasitás a +! (n cim ---) szó. Ez a ! utasitással megegyező bemenő paramétereket vár, azonban a 16 bites adatot a cimen lévő 16 bites adathoz hozzáadja, és az összeget helyezi el a cim...cim+1-re.

Az alábbi táblázatban a memóriairó és memóriaolvasó utasitásokat foglaltuk össze (n,n1,n2 16-bites, b,b1,b2 8-bites, d,d1,d2 32-bites adatot jelent, a cim pedig a memóriacimet jelöli).

szó	adatverem előtt	adatverem után	mem.tart. előtt	mem.tart. után
@	cim	n	n	n
C@	cim	b	b	b
2@	cim	d	d	d
!	n1 cim	-	n2	n1
+!	n1 cim	-	n2	n1+n2
C!	b1 cim	-	b2	b1
2!	d1 cim	-	d2	d1

Lássunk egy példát az új utasitások használatára. A 20000-es cimre tegyünk le 100-at, majd +!-lel adjunk hozzá 11-et, és a cim új tartalmát irassuk ki:

100 20000 ! 11 20000 +! 20000 @ . 111 ok

2.3.1.2. Egyéb memóriakezelő műveletek

Elsőként egy speciális utasitással ismerkedünk meg: **TOGGLE** (cim n ---). Ez a cimen található 8 bites adatot kizáró vagy (XOR, l. 2.ó.2.) kapcsolatba hozza n alsó 8 bitjével, és ezt az értéket a cimre helyezi.

Memóriaterületek áthelyezésére szolgál a **MOVE** és **CMOVE** utasitás (ciml cim2 n ---). Mindkettő bemenő paraméterei: a verem 3. szintjén az áthelyezendő memóriatartomány kezdőcime (ciml), a 2. szinten a cél memóriatartomány kezdőcime (cim2), és a verem tetején az áthelyezendő bájtok száma (n). A MOVE kétbájtos egységenként (16 bitenként) másol, mig a CMOVE ezt bájtonként (8 bitenként) hajtja végre. Fontos tudnivaló, hogy mindkét utasitás növekvő cimsorrendben működik.

Egy memóriaterület nullákkal való feltöltését végzi a **ERASE** utasitás (cim n ---). A verem 2. szintjén a memóriarész kezdőcimét, az 1. szinten pedig a nullázandó bájtok számát várja.

Egy memóriaterület szóköz karakterekkel (ASCII 32) való feltöltését végzi a **BLANK** vagy **BLANKS** utasitás (cim n ---). A verem 2. szintjén a memóriarész kezdőcimét, az 1. szinten pedig a kitöltendő bájtok számát várja.

A memória adott részét tölti fel bizonyos tartalommal a **FILL** (cim n b ---). A verem 3. szintjén kell elhelyezni a kitöltendő memória első cimét, fölötte a bájtok számát,s végül a verem felső szintjén a kivánt tartalmat (b). Például töltsük ki a 20000-től 20053-ig terjedő memóriatartományt 11-gyel:

20000 54 11 FILL ok

Végül a memória bizonyos tartományának tartalmát jelenithetjük meg a képernyőn a **DUMP** utasitás segitségével (cim n ---). A verem 2. szintjén kell megadni a tartomány kezdőcimét, az 1. szinten pedig a megjelenitendő bájtok számát (n).

2.3.2. Konstansok

Gyakran előfordul, hogy bizonyos számokat többször használunk egy programban. Ezeket a könnyebb áttekinthetőség kedvéért célszerű konstansként (állandó) definiálni. A későbbi programmódositásoknál igen jó szolgálatot tesz ez a módszer.

Konstanst a FORTH-ban a **CONSTANT** szó segitségével definiálhatunk (n ---). Bemenő paramétere a verem tetején levő szám:

```
n CONSTANT akármi
```

ahol n egy egész szám, akármi pedig a felhasználó által szabadon megválasztott azonositó. Például:

3 CONSTANT HAROM ok

Ezután ha leirjuk a HAROM azonositót, a verem tetejére kerül a 3 szám:

HAROM . 3 ok

Dupla hosszú egész tipusú konstanst a fentiekhez hasonlóan definiálhatunk a **2CONSTANT** szó felhasználásával (d ---):

d 2CONSTANT akármi

ahol d egy duplapontos egész szám, akármi pedig a felhasználó által szabadon megválasztott azonositó.

2.3.3. Változók

Ha bizonyos adatokat ideiglenesen tárolni szeretnénk, ehhez tárolórekeszeket, változókat kell definiálnunk. A FORTH-ban erre szolgál a **VARIABLE** utasitás, amely a 3.2.1.2 pontban tárgyalandó DP rendszerváltozó által mutatott cimtől kezdődően lefoglal két memóriarekeszt az adatok tárolására. Használata a következő:

VARIABLE akármi

vagy a fig-FORTH-ban:

n VARIABLE akármi

ahol n egy egész szám, akármi pedig a felhasználó által megválasztott azonositó. Az első verzióban a változó kezdeti értéke határozatlan, a második esetben pedig n. Például:

VARIABLE CITROM ok

vagy a fig-FORTH-ban:

4 VARIABLE CITROM ok

Ezután ha leirjuk a CITROM azonositót, a verem tetejére kerül a változó számára lefoglalt memória cime. Igy a változó értékét a 2.3.1 pontban megismert @ utasitással vehetjük elő, ill. a ! vagy +! utasitással változtathatjuk meg. Például

először legyen 3 citromunk, majd ehhez vegyünk még öt darabot:

```
3 CITROM ! 5 CITROM +! CITROM @ . 8 ok
```

Duplapontos egész tipusú változót a **2VARIABLE** szó segitségével definiálhatunk:

```
2VARIABLE akármi
```

vagy a fig-FORTH-ban:

```
d 2VARIABLE akármi
```

ahol d a változó kezdeti értéke, akármi pedig a felhasználó által megválasztott azonositó. Az igy deklarált változók értékéhez a 2@ szó segitségével férhetünk hozzá, megváltoztatni pedig a 2!-lel tudjuk.

Speciális esetben használjuk a felhasználói (= user) változót. Akkor lehet rá szükség, ha a rendszer ROM-ban (Read Only Memory = csak olvasható memória) van. Mivel a VARIABLE a FORTH rendszerhez illeszti a változót, ilyenkor ennek értékét nem lehet megváltoztatni. A felhasználói változók a visszatérési verem fölé kerülnek (l. az F.6 függeléket), amely mindig RAM-ban van (Random Access Memory = irható/olvasható memória). Felhasználói változót a **USER** utasitással hozhatunk létre:

```
offs USER akármi
```

ahol akármi a definiálandó változó neve, offs pedig a változóhoz tartozó memóriát adja meg, éspedig a felhasználói mutatóhoz (= User Pointer, UP) viszonyitott relativ cim (eltolás) formájában. A felhasználói mutató a felhasználói változók tárolására kijelölt memóriaterület kezdetének cimére

mutat, értéke az R0 változóban is megtalálható (l. az F6. függeléket).

2.4. Veremkezelő műveletek

2.4.1. Adatvermet kezelő műveletek

A 2.1 szakaszban már megismertük a FORTH programozási nyelvnek azt a sajátosságát, amely szerint a FORTH az elvégzendő műveletek operandusait általában nem a hagyományos programozási nyelvekben (FORTRAN, BASIC, PASCAL) megszokott fiókosszekrényre emlékeztető elrendezésben, változókban tartja, hanem egy LIFO-elven működő adattároló veremben. Ebből fontos következtetést vonhatunk le: szükségünk lesz olyan műveletekre, amelyek az adatveremben tárolt adatok sorrendjét és/vagy számát képesek megváltoztatni.

Lássunk egy példát! Szeretnénk kiszámitani 15^2 (= 15*15) értékét, de az adatveremben csak egy darab 15-ös szám áll rendelkezésre. A verem tetején levő szám megkettőzésére szolgál a **DUP** utasitás (n --- n n):

```
15 DUP * . 225 ok
```

Egyelőre magyarázat nélkül fogadjuk el, hogy a * szoroz és a . kiir. A **?DUP** vagy **-DUP** csak akkor duplázza meg a verem legfelső szintjét, ha az ott található szám nem nulla. Például:

```
1 ?DUP . . 1 1 ok
0 ?DUP . . 0 Error: "." empty stack ok
```

Látható, hogy a második esetben hibajelzést kaptunk.

Ennek oka a második . utasitás, ugyanis a ?DUP nem kettőzte meg a nullát, igy a verem üres volt.

A másik olyan veremművelet, amely egy veremszintet érint, a **DROP** (n ---). Végrehajtása során az adatverem legfelső elemét törli.

A **SWAP** utasitás (n1 n2 --- n2 n1) már két veremszintet igényel. Pontos definiciója a következő: az adatverem két legfelső elemének sorrendjét felcseréli. Jobb megértéséhez gépeljük be a következő két utasitássort:

```
1 32767 . . 32767 1 ok
1 32767 SWAP . . 1 32767 ok
```

Figyeljük meg, hogy ha számokat iratunk ki egymás utáni . utasitásokkal, akkor - mivel először a verem tetején levő íródik ki, majd a következő, és igy tovább - , éppen forditott sorrendben jelenik meg a szokásos jelöléshez képest. A veremszintek jelölésénél ugyanis a legjobboldalibb a verem teteje, kiiratáskor pedig balról az első.

Az **OVER** utasitás (n1 n2 --- n1 n2 n1) a verem második szintjén levő elemet még egyszer elhelyezi a verem tetejére. Hatását az alábbi példával érzékeltethetjük.

```
1 2 OVER . . . 1 2 1 ok
```

Az egyetlen háromoperandusú veremkezelő művelet a **ROT** (n1 n2 n3 --- n2 n3 n1). Már a neve is sejteti, hogy itt valamiféle rotációról van szó. Ez a sejtésünk be is igazolódik ha a definiciót tekintjük: a ROT utasitás az adatvermen levő adatok sorrendjét változtatja meg oly módon, hogy a harmadik szinten levő elem kerül legfelülre, és a többi elem egy szinttel lejjeb tolódik .

Vegyük észre, hogy ez ciklikus művelet, hiszen a ROT

ROT ROT sorozat visszaállitja az eredeti sorrendet.

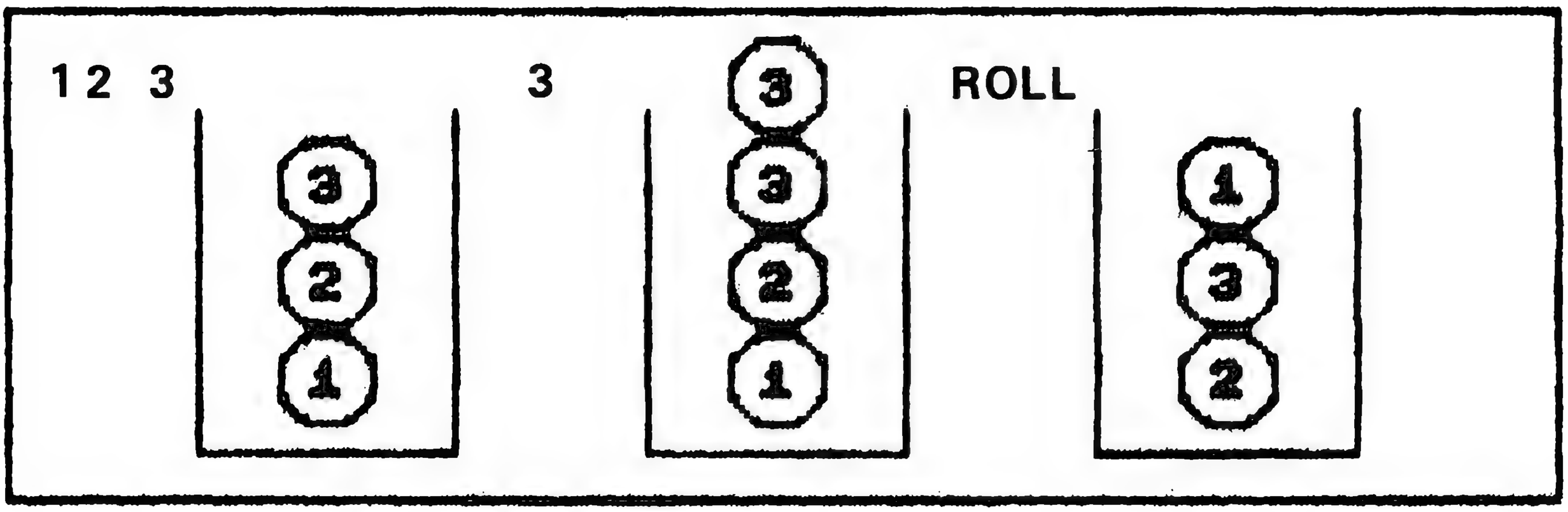
Foglaljuk össze a már megismert veremkezelő műveleteket egy táblázatba (n, n1, n2, n3 egy-egy egész számot jelöl):

művelet	előtt	után	megjegyzés
DUP	n	n n	
?DUP	n	n n	ha n<>0
	n	n	ha n=0
DROP	n	-	
SWAP	n1 n2	n2 n1	
OVER	n1 n2	n1 n2 n1	
ROT	n1 n2 n3	n2 n3 n1	

A következő két utasitás része a FORTH-79-nek, de a fig-FORTH rendszerek általában nem tartalmazzák őket.

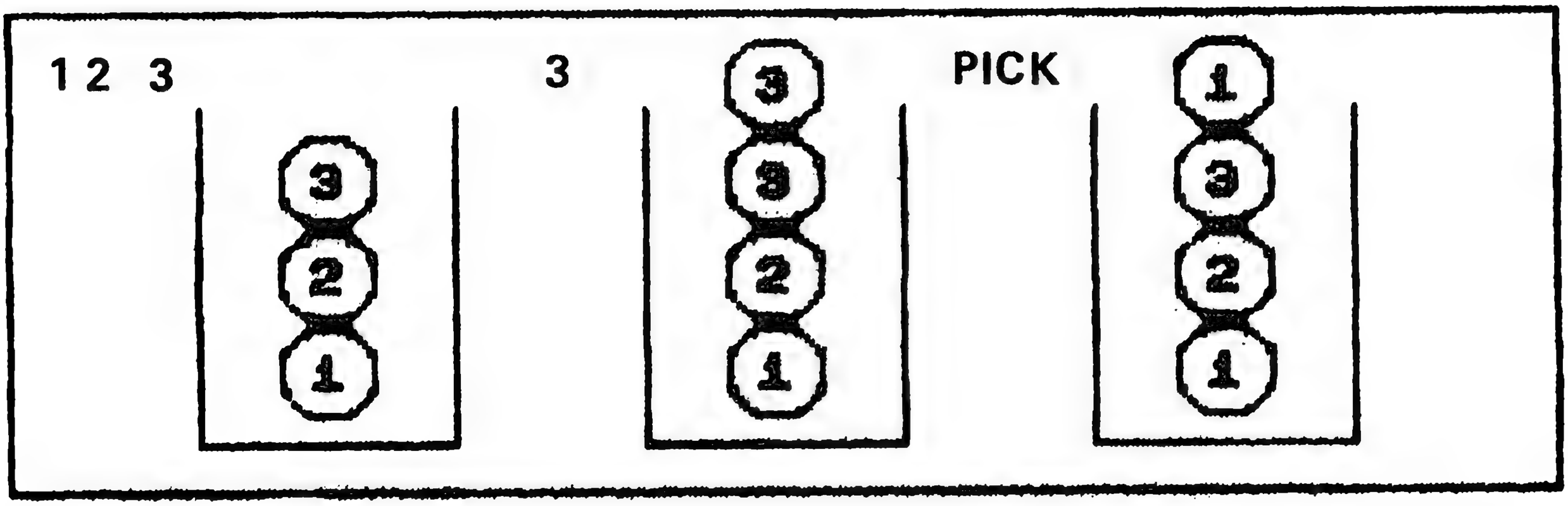
Első ilyen utasitás a **ROLL**, amely tulajdonképpen a ROT egy általánositott és ezért jóval szélesebb körben használható változata. Egy alapvető dologban különbözik az eddig ismertetett veremműveletektől, mégpedig abban, hogy egy plusz bemenő paramétert igényel amely megmondja, hogy milyen mélyen kell kezdeni a "ROLL-ozást". Tehát n ROLL azt jelenti: az adatverem n-edik szintje kerül a legfelső szintre, és az összes többi egy szinttel lejjebb csúszik (2.7. ábra) (ROT = 3 ROLL). Lássunk egy példát:

```
1 2 3 ROT . . . 1 3 2 ok
1 2 3 3 ROLL . . . 1 3 2 ok
```



2.7. ábra
A verem állapotai a 3 ROLL végrehajtása során

A csoport másik tagja a **PICK**. Ismét nagyon szemléletes utasitásnévvel állunk szemben. Az angol nyelvben a to pick ige ugyanis azt jelenti: felvesz, felszed, felcsip. Nos a PICK utasitás leveszi az adatverem tetején levő számot, és az ennek megfelelő mélységű veremszinten elhelyezett adatot a verem tetejére teszi (OVER = 2 PICK), miközben az eredeti adat sem vész el (2.8. ábra).



2.8. ábra
A verem állapotai a 3 PICK előtt és után

Ha az Olvasó FORTH rendszerében nem szerepel a fenti két utasitás, megvalósithatja a 3.7. szakasz 3.7.1. és 3.7.2. feladatainak felhasználásával.

Végül ismerkedjünk meg a <u>veremmutatóval</u> (= Stack Pointer). Ez egy olyan speciális változó, amely a memóriában mutatja a verem aktuális poziciójá t. Ennek értékét az **SP@** utasitás adja meg (--- n). Ez az érték a verem tetején található elem cimét jelenti. A verem átalában úgy van megszervezve, hogy a memóriában "lefelé" nő, vagyis a magasabb veremszintekhez kisebb memóriacim, SP@ érték tartozik. Az alapértéket (amikor a verem üres) az **S0** vagy **TIB** változóból olvashatjuk ki. Használatát egy hasznos utasitássorozat segitségével mutatjuk be, amely a verem szintjeinek számát adja meg. Ez azt jelenti, hogy hány egész szám van a vermen:

```
SP@     ( ... sp=aktuális érték )
S0 @    ( ... sp sp0=alapérték )
SWAP    ( ... sp0 sp )
```

```
-       ( ... n=sp0-sp=bájtok száma )
2 /     ( ... m=n/2=szintek száma )
```

Ezt az utasitássorozatot pótolja a FORTH-79-ben a **DEPTH** szó (--- m, a verem mélysége).

A veremmutató értékét alapállapotba lehet hozni, vagyis a vermet törölhetjük az **SP!** szóval (l. a 3.6.1.).

A FORTH-79 szabvány tartalmaz duplapontosságú (32 bites) egész aritmetikát is. Az ilyen, duplapontos egész tipusú adatokon végzett műveletek támogatására kerültek a szabványba a duplapontos veremszinteket kezelő műveletek. Ezek működése értelemszerűen azonos egyszeres hosszú megfelelőikkel, azzal a különbséggel, hogy itt a 32 bites adatok forgalma zajlik. Ezért csak táblázatos formában ismertetjük őket (d, d1, d2 egy-egy duplapontos egész számot jelöl):

	Az adatverem		
művelet	előtt	után	megjegyzés
2DUP	d	d d	= OVER OVER
2DROP	d	-	= DROP DROP
2SWAP	d1 d2	d2 d1	= 4 ROLL 4 ROLL
2OVER	d1 d2	d1 d2 d1	= 4 PICK 4 PICK

2.4.2. Visszatérési vermet kezelő műveletek

A 2.1.1. szakaszból kiderült, a FORTH-ban az adatvermen kivül van egy másik, ún. visszatérési verem is. Bár a felhasználó hozzáférhet a visszatérési verem tartalmához, de ez rendkivüli körültekintést igényel, ugyanis a FORTH ezt a vermet rendszerinformációk tárolására használja, és ezek felülirása "életveszélyes".

Az utasitások visszatérési vermet kezelő csoportjának első tagja a **>R** (n ---). Az adatverem tetejéről levesz egy számot, és azt a visszatérési verem legfelső szintjére helyezi. Az adatveremből az adat elvész.

A >R komplemense az **R>** (--- n). Ez az utasitás a visszatérési verem tetejéről tesz egy számot az adatverem legfelső szintjére.

Az **R@** vagy **R** (--- n) utasitás az R> -hoz hasonlóan működik azzal az eltéréssel, hogy a visszatérési verem tetején érintetlen marad a szám.

Az **I, I'** és **J** utasitások rendre a visszatérési verem első, második, harmadik szintjét helyezik az adatveremre. A visszatérési veremben tárolt adatok nem károsodnak. Az alábbi táblázatban a visszatérési vermet kezelő műveleteket foglaltuk össze (n, n1, n2, n3 egy-egy 16 bites egész számot jelöl).

2.4. Veremkezelő műveletek

	adat	visszatérési	adat	visszatérési
művelet	előtt		után	
>R	n	-	-	n
R>	-	n	n	-
R@	-	n	n	n
I	-	n1 n2 n3	n3	n1 n2 n3
I'	-	n1 n2 n3	n2	n1 n2 n3
J	-	n1 n2 n3	n1	n1 n2 n3

Legfontosabb felhasználási területük a visszatérési verem átmeneti adattárolásra való felhasználása. Példákat a 3.3.2.2. pontban láthatunk majd.

A veremmutatóhoz hasonlóan létezik egy <u>visszatérési veremmutató</u> is (= Return stack Pointer). Ez egy olyan speciális változó, amely a memóriában mutatja a visszatérési verem aktuális poziciójá t. Ennek értékét az **RP@** utasitás adja meg (--- n).

A visszatérési veremmutató értékét alapállapotba lehet hozni, vagyis a visszatérési vermet törölhetjük az **RP!** szóval (l. a 3.6.1.). Az alapértéket az **R0** rendszerváltozó hordozza.

2.5. Aritmetikai utasitások

Amennyiben felmérést készitenénk különböző magas szintű programozási nyelvek forrásszövegű listáiban, úgy az értékadás művelete mellett bizonyára az aritmetikai operátorokat találnánk leggyakrabban. Mivel a FORTH-ban az adatforgalom a vermen keresztül zajlik, és az értékadásnak

megfeleltethető ! műveletre ritkábban van szükség, ezért valószinűleg még gyakoribbak az aritmetikai műveletek.

Az operátorok ezen osztályán belül többféle felosztás is elképzelhető:

- az operandusok jellege (egész, duplapontos, vegyes)
- az operandusok száma (1,2,3) szerint.

Ugy véljük az első felosztás könnyebb megértést tesz lehetővé, ezért a továbbiakban ezt alkalmazzuk.

2.5.1. Egész tipusú műveletek

A FORTH a négy alapműveletet a matematikában már jól megszokott szimbólumokkal jelöli. Az összes aritmetikai művelet bemenő adatát az adatveremről veszi, és az eredményt is oda helyezi vissza. Itt szeretnénk felhivni a figyelmet az operandusok helyes sorrendjének betartására (kivonás, osztás esetén)! Az alábbi táblázat összefoglalja az alapműveleteket (n, n1, n2 egész számokat jelölnek).

	az adatverem		
művelet	előtt	után	megjegyzés
+	n1 n2	n=n1+n2	
-	n1 n2	n=n1-n2	
*	n1 n2	n=n1*n2	
/	n1 n2	n=n1/n2	(egészosztás)

2.5. Aritmetikai utasitások

A FORTH rendszerek túlnyomó többsége nem törődik az aritmetikai műveletek során esetleg fellépő túlcsordulással. Más programozási nyelvekben például +32767-hez (a legnagyobb pozitiv szám, ami 16 biten, kettes komplemensen ábrázolható) egyet hozzáadunk, akkor hibajelzést kapunk. A FORTH-nál más a helyzet.

```
32767 1+ . -32768 ok
```

A FORTH tehát elvégzi a műveletet, és a túlcsordulást figyelmen kivül hagyja. Emiatt gyakran furcsa eredményeket kapunk.

A most következő műveletek ugyan nem feltétlenül szükségesek, de meglétük pontossági vagy sebességi követelményekkel indokolható, és persze a felhasználó kényelme sem utolsó szempont.

Ha két számot elosztunk egymással, előfordulhat olyan eset, hogy nem a hányados, hanem a maradék érdekes. Ennek előállitására szolgál a **MOD** utasitás (n1 n2 --- n3=[n1 mod n2]). Egyenértékű a következő utasitássorozattal:

```
2DUP / * -
```

Első pillantásra nemigen látszik, hogy ez igaz, ezért lássuk részletesebben:

```
          ( n1 n2 )
2DUP      ( n1 n2 n1 n2 )
/         ( n1 n2 [n1/n2] )
*         ( n1 [n1/n2]*n2 )
-         ( n1-[n1/n2]*n2 = n1 mod n2 )
```

A MOD továbbfejlesztett változata a **/MOD** (n1 n2 --- n3 n4, n3=[n1 mod n2] n4=[n1/n2]). Ez az utasitás nemcsak a maradékot, hanem a hányadost is az adatveremre helyezi, mégpedig a verem második szintjére a maradékot fölé pedig a hányadost. A következő utasitássorozat egyenértékűségének belátását az olvasóra bizzuk:

```
/MOD = 2DUP / DUP >R * - R>
```

A 16 biten ábrázolható számok értéktartománya a -32768...+32767 intervallum. Előfordulhatnak olyan esetek amikor egy közbenső eredmény túl nagy, de a számitások általában nem teszik szükségessé a duplapontosságú aritmetika használatát. Ilyen esetben hasznosak lehetnek a ***/** (n1 n2 n3 --- n4=[n1*n2] mod n3) és a ***/MOD** (n1 n2 n3 --- n4 n5, n4=[n1*n2] mod n3 n5=[n1*n2]/n3)utasitások:

	Az adatverem		
művelet	előtt	után	megjegyzés
*/	n1 n2 n3	n	n=(n1*n2)/n3
*/MOD	n1 n2 n3	n4 n5	n4=(n1*n2) mod n3
			n5=(n1*n2)/n3

Miben különbözik például a */ a * /-től? A magyarázat egyszerű: a */ a szorzás elvégzése után az eredményt 32 biten ábrázolja, ami azt jelenti, hogy a szám akár 2^{31} is lehet,

mig a * / szekvencia esetén az esetleges átvitelek figyelmen kivül maradnak. Ugyanezek vonatkoznak értelemszerűen a */MOD-ra is.

Lássunk egy példát! Egy kezdő villamosmérnök fizetése 3800 Ft. Ennek 43%-át levonják gyermektartás, szakszervezeti tagdij, SZTK stb. miatt. Mérnökünk kiváncsi a levonás összegére.

```
3800 43 100 */ . 1634 ok
3800 43 * 100 / . 323 ok
```

A különbség szemmel látható, a második megoldás a túlcsordulás miatt hibás.

Van néhány egyszerű, külön magyarázatot nem igénylő aritmetikai utasitás is, amelyeket a következő táblázatban ismertetünk. Ezek mind az adatverem legfelső egy vagy két szintjén végeznek műveletet (n, n1, n2 16 bites egész számot jelölnek).

művelet	Az adatverem előtt	Az adatverem után	megjegyzés
1+	n	n+1	
1-	n	n-1	
2+	n	n+2	
2-	n	n-2	
2*	n	n*2	
2/	n	n/2	
ABS	n	n	
MINUS	n	-n	
NEGATE	n	-n	NEGATE=MINUS
+-	n1 n2	-n1	ha n2<0
	n1 n2	n1	ha n2>=0
MIN	n1 n2	n=min(n1,n2)	
MAX	n1 n2	n=max(n1,n2)	

2.5.2. Duplapontosságú aritmetika

Az egyszeres pontosságú utasitásokhoz képest mindössze annyi a különbség, hogy itt mind a bemenő, mind a kimenő adatok 32 biten ábrázolt előjeles számok, amelyek tehát két veremszintet foglalnak el.

művelet	Az adatverem előtt	után	megjegyzés
D+	d1 d2	d1+d2	
D-	d1 d2	d1-d2	
DABS	d1	d1	
DMINUS	d1	-d1	
DNEGATE	d1	-d1	DNEGATE=DMINUS
D+-	d1 d2	-d1	ha d2<0
	d1 d2	d1	ha d2>=0
DMIN	d1 d2	min(d1,d2)	
DMAX	d1 d2	max(d1,d2)	

2.5.3. Vegyes hosszúságú aritmetika

Annak számára, aki idáig eljutott, bizonyára nem meglepő az a tény, hogy léteznek olyan FORTH utasitások, amelyek 16 és 32 bites bemenő adatokat egyaránt felhasználnak.

Az M+ (d n --- d=n+d) az adatverem tetején levő 16 bites számot hozzáadja az alatta levő 32 bites számhoz. Az eredmény 32 bites előjeles szám.

Az M* (n1 n2 --- d=n1*n2) utasitás két 16 bites előjeles számot szoroz össze. Az eredményként képződő 32 bites előjeles számot az adatveremre helyezi.

Az M/ (d n --- n1 n2, n1=[d mod n] n2=d/n) definiciója a következő: elosztja az adatverem tetején levő 16 bites számmal az alatta elhelyezkedő 32 bites számot. Az

eredmény 16 bites szám lesz.

Az **M*/** (d n1 n2 --- d1=[n1*d]/n2) utasitás az adatverem harmadik szintjén levő 32 bites és a második szinten található 16 bites számot összeszorozza. Az igy képződött 48 bites szorzatot elosztja az adatvermen legfelül elhelyezett 16 bites számmal. A végeredmény 32 bites előjeles szám lesz.

Az **M/MOD** (du1 u1 --- u2 du2) a verem tetején levő előjel nélküli számmal elosztja az alatta levő duplapontos előjel nélküli számot. Eredményként a verem 2. szintjére a 16 bites maradék, fölé pedig a duplapontos hányados kerül.

Az **U/** (du u1 --- u2 u3, u2=[du mod u1] u3=du/u1) utasitás a verem tetején levő előjel nélküli számmal elosztja az alatta levő duplapontos, előjel nélküli számot. Eredményként a verem 2. szintjére a 16 bites maradék, fölé pedig a 16 bites hányados kerül.

Az **U*** (u1 u2 --- du=u1*u2) utasitás a verem tetején levő két előjel nélküli egész számot összeszorozza, és az eredményt 32 bites előjel nélküli számként a veremre helyezi.

Az alábbi táblázat összefoglalja a változó hosszúságú aritmetikai utasitásokat (n1, n2 16 bites előjeles, u1, u2, u3 16 bites előjel nélküli, d 32 bites előjeles, du, du1, du2 32 bites előjel nélküli egész szám).

2.5. Aritmetikai utasitások

	Az adatverem		
művelet	előtt	után	megjegyzés
M+	n d	d1=n+d	
M*	n1 n2	d=n1*n2	
M/	d n	n1 n2	n1=d mod n
			n2=d/n
M*/	d n1 n2	d1=n1*d/n2	
M/MOD	du1 u1	u2 du2	u2=du1 mod u1
			du2=du1/u1
U/	du u1	u2 u3	u2=du mod u1
			u3=du/u1
U*	u1 u2	du=u1*u2	

2.6. Relációk és logikai műveletek

A relációk olyan műveletek, amelyek bizonyos feltételek meglététől vagy hiányától függően logikai értékeket szolgáltatnak. A logikai értékek kétfélék lehetnek: igazak, ha a feltétel teljesült, és hamisak, ha nem. Precizen megfogalmazva a relációk olyan függvények, amelyek értelmezési tartománya a 16 vagy 32 bites egész számokból alkotott párok halmaza, értékkészlete pedig a logikai (Boolean) értékek halmaza. A 0 a hamis (false), a nem nulla az igaz (true). A FORTH itt is kétféle relációt kinál, az egész és a duplapontos egész számokra vonatkozóakat, továbbá itt is lehetséges a megkülönböztetés az argumentumok száma szerint. Az összes relációs művelet a megfelelő számú bemenő paraméterét levészi a veremről. Egyargumentumú reláció esetén a pár másik eleme zérus, amint ez a jelölésmódból is világosan kitűnik.

2.6.1. Relációk

A relációk igaz értéke általában -1, a hamis 0. Az előjel nélküli egészeket u, u1, ..., az előjeles egészeket n, n1, ..., a duplapontos előjel nélküli egészeket du, du1, ..., duplapontos előjeles egészeket d, d1, ... jelöli. Az egész számokra vonatkozó relációk a következők:

	Adatverem		
művelet	előtt	után	megjegyzés
=	n1 n2	-1	ha n1=n2
	n1 n2	0	ha n1<>n2
<	n1 n2	-1	ha n1<n2
	n1 n2	0	ha n1>=n2
>	n1 n2	-1	ha n1>n2
	n1 n2	0	ha n1<=n2
U<	u1 u2	-1	ha u1<u2
	u1 u2	0	ha u1>=u2
0=	n	-1	ha n=0
	n	0	ha n<>0
0<	n	-1	ha n<0
	n	0	ha n>=0
0>	n	-1	ha n>0
	n	0	ha n<=0
D=	d1 d2	-1	ha d1=d2
	d1 d2	0	ha d1<>d2
D0=	d	-1	ha d=0
	d	0	ha d<>0
D<	d1 d2	-1	ha d1<d2
	d1 d2	0	ha d1>=d2
DU<	du1 du2	-1	ha du1<du2
	du1 du2	0	ha du1>=du2

2.6.2. Logikai műveletek

A logikai műveletek valójában a kettes számrendszer aritmetikai operációi. Alapvetően egybites operandusokra szokták értelmezni őket. A FORTH-ban ezt oly módon

terjesztették ki, hogy a 16 bites számok minden bitjére végrehajtják a megfelelő műveletet. A FORTH-79 három logikai utasitást tartalmaz, amelyek a következők:

AND (n1 n2 --- n3)
a vermen levő két 16 bites szám minden megfelelő bitpárjára végrehajtja a logikai ÉS műveletet, amelynek igazságtáblája a következő:

```
0 0 --- 0
0 1 --- 0
1 0 --- 0
1 1 --- 1
```

OR (n1 n2 --- n3)
a vermen levő két 16 bites szám minden megfelelő bitpárjára végrehajtja a logikai VAGY műveletet, amelynek igazságtáblája a következő:

```
0 0 --- 0
0 1 --- 1
1 0 --- 1
1 1 --- 1
```

NOT (n1 --- n2)
a vermen levő 16 bites szám minden bitjére végrehajtja a logikai NEM műveletet, amelynek igazságtáblája a következő:

```
0 --- 1
1 --- 0
```

például: -1 NOT . 0 ok
0 NOT . -1 ok

A fig-FORTH-ban nem szerepel a NOT, de ha logikai

feltétel képzéséhez akarjuk használni, helyettesithetjük a 0= relációval. Például 0= NOT megfelel a 0= 0= formulának (jelentése: nem zérus).

A **XOR** (n1 n2 --- n3)
a fig-FORTH operátora. A vermen levő két 16 bites szám minden megfelelő bitpárjára végrehajtja a logikai KIZÁRÓ VAGY műveletet, amelynek igazságtáblája a következő:

0 0 --- 0
0 1 --- 1
1 0 --- 1
1 1 --- 0

2.7. Adatkonverziós műveletek

Amint ezt a 2.2. részben már láthattuk, a FORTH-79 szabvány három adattipust ismer (egész, duplapontos egész, szöveg). Ezen tipusok között lehetséges az átalakitás. Ezeket a konverziós műveleteket ismertetjük az alábbiakban.

2.7.1. Konverzió egészből duplapontos egészbe

Az erre szolgáló utasitás neve: **S->D** (n --- d). Végrehajtása során a verem tetején levő 16 bites szám előjelét oly módon terjeszti ki, hogy alkalmassá váljon a duplapontosságú aritmetika bemeneti adataként való felhasználására. A szám veremben elfoglalt poziciója nem változik.

2.7.2. Konverzió számok és karakterfüzérek között

A FORTH 2-től 36-ig bármely számrendszerben képes dolgozni, és a kivánt számrendszer beállitását a számrendszer alapszámának **BASE** nevű rendszerváltozóba töltésével érhetjük el (az alapértelmezés 10). Az adatkonverziók végrehajtása előtt ennek beállitásával tisztázni kell, hogy milyen számrendszerben óhajtunk dolgozni.

```
16 BASE ! ok      ( hexadecimális )
8 BASE ! ok       ( oktális )
```

A BASE rendszerváltozó beállitására egyszerűbb eszközöket is kinál a FORTH az alábbi utasitások formájában:

```
HEX      = 16 BASE !
DECIMAL  = 10 BASE !
```

2.7.2.1. Számok átalakitása karakterfüzérré

E konverzió célja általában az olvasható kiirás (például táblázatok készitése). Erre elsősorban a 2.9.1. pontban ismertetendő formátumvezérlő utasitások használhatók, a karakterfüzérré való átalakitást csak különleges igények esetén használjuk.

Az átalakitás közvetlenül csak duplapontos egész számokra használható, kis trükkel azonban szimpla egészekre is értelmezhető. Erre a fejezet végén visszatérünk.

Lássunk egy példát! Menetrendi táblázatban az indulási időt az alábbi formában kell kiirni:

óó:pp

ahol óó az óra, pp a perc két-két számjegyét jelöli, a kettőspont pedig az elválasztójel. Szeretnénk elérni az alábbi futási képet:

12.50 (utasitások) 12:50 ok

A számban levő pont jelzi, hogy az duplapontos. Az alábbiakban ismertetjük a konverzió lépéseit, azaz az "utasitások" kifejtését.

Az utasitáscsoport első tagja a <# . Előkésziti a szám átalakitását. A karakterfüzért a szövegpufferbe teszi, amelynek kezdőcimét a PAD utasitás adja meg.

A # utasitás a BASE figyelembevételével átalakitja a szám egy jegyét karakterré és elhelyezi a karakterfüzérben. Ha a jegy nem értékes, úgy nullát ir be a füzérbe. Jegyezzük meg, hogy a konverzió a legkisebb helyiértéktől halad a magasabb helyiértékek felé, tehát példánkban az első # a nullát alakitja át.

A **HOLD**-az adatverem tetején levő számot ASCII kódként értelmezi, és belehelyezi a kimenő karakterfüzér aktuális pozicijába .

A **#S** mindaddig folytatja az átalakitást, mig értékes jegy képződik. Az értéktelen 0-kat elhagyja.

A konverziót a #> utasitás zárja le. Az eredményfüzér cimét és hosszát ebben a sorrendben az adatveremre helyezi.

Az eddigiek alapján már meg tudjuk oldani a fejezet elején bemutatott példát. Tudjuk, hogy a kettőspont ASCII kódja 58.

```
12.50     ( --- 1250 )
<#        ( konverzió start )
#         ( "0" ---> PAD )
#         ( "5" ---> PAD )
58 HOLD   ( ":" ---> PAD )
#S        ( "2" ---> PAD "1" ---> PAD )
#>        ( --- PAD 5 )
TYPE 12:50 ok
```

Mint látható, a konverzió után az eredményfüzér kezdőcime (PAD) és hossza (5) a veremre került. (A TYPE utasitást a 2.9. szakaszban ismertetjük, működése: kiirja a karakterfüzért.)

A **SIGN** utasitást általában közvetlenül a #> előtt szokás alkalmazni. Hatására a minuszjel (-) kódja kerül a kimenő karakterfüzérbe abban az esetben, ha az adatverem harmadik szintje negativ, ellenkező esetben hatástalan.

A fentebb ismertetett utasitásokat a következőképpen használjuk: <# utasitássor #> ahol az utasitássor bármilyen FORTH utasitást tartalmazhat (általában #,#S,HOLD használatos). Ebben a formában előjel nélküli duplapontos számok formázására használható. Ha 16 bites számot akarunk átalakitani, úgy előzetesen egy nullát kell a verem tetejére helyeznünk.

A <# utasitássor SIGN #> szekvencia előjeles duplapontos számok átalakitására használható. Ekkor az adatvermen a szám abszolút értéke alatt ott kell lenni a szám felső két bájtjának, hogy a SIGN helyesen működjék. Ezt a következő utasitássorozattal készithetjük elő:

SWAP OVER DABS

Szimpla előjeles egész számok esetén a veremszintek a következőképpen alakulnak: legalul a szám, felette a szám

abszolút értéke, legfelül egy 0.

Foglaljuk táblázatba a számok karakterfüzérré való átalakitására használható eljárásokat (n 16 bites egész, d 32 bites egész, fd d felső 16 bitje).

eljárás	bemenő	kimenő
<# utasitássor #>	d	cim hossz
<# utasitássor #>	n 0	cim hossz
<# utasitássor SIGN #>	fd d	cim hossz
<# utasitássor SIGN #>	n n 0	cim hossz

Végezetül tekintsünk néhány példát:

1. példa:
Irassunk ki egy számot olyan formában, hogy minden számjegyet egy szóköz válasszon el a másiktól! Tudjuk, hogy a szám 4-jegyű, és a szóköz ASCII kódjára a **BL** konstanssal lehet hivatkozni.

```
123.4 ok ( a pont jelzi: a szám duplapontos)
<# # BL HOLD # BL HOLD # BL HOLD #S BL HOLD #> ok
TYPE 1 2 3 4 ok
```

2. példa:
Megbizást kaptunk árukódszámok kiirását végző program készitésére. A kért formátum: j-jjjj-jjjj, ahol j egy-egy számjegyet jelöl. A kötőjel ASCII kódja 45.

```
16004028.3 ok
```

<# # # # # 45 HOLD # # # # # 45 HOLD #S #> TYPE 1-6004-0283 ok

3. példa:
8-jegyü negativ számokat kell kiiratni a következő formában:
jj,jjj,jjj. A vessző ASCII kódja 44.

-1234567.8 SWAP OVER DABS ok
<# # # # 44 HOLD # # # # 44 HOLD #S SIGN #> ok
TYPE -12,345,678 ok

2.7.2.2. Konverzió karakterfüzérből számba

Az első konverziós utasitás a **NUMBER** (cim --- d). Bemenő adata az átalakitandó, szóközzel lezárt karakterfüzér cime. Az átalakitást cim+1-től végzi! A füzér elején levő előjelet figyelembe veszi, . Ha a füzér tartalmazza a pont karaktert, ennek pozicióját a **DPL** változóban adja meg, egyébként DPL értéke -1. Tekintsünk egy példát, amelyben egy duplapontos szám utolsó jegye elé beszúrunk egy nullát! Ehhez a karakterfüzérré való átalakitást használjuk fel. A konverzió után a számot a NUMBER segitségével visszaalakitjuk, és a D.-tal kiiratjuk (a D. a . duplapontos megfelelője, l. a 2.9.1. pontot). A nulla ASCII kódja 48, a szóközét a BL konstans adja meg.

```
123.456                          ( 123456 )
<# BL HOLD # 48 HOLD #S #>       ( PAD 8 )
DROP                             ( PAD )
1-                               ( PAD-1 )
NUMBER                           ( 1234506 )
D. 1234506 ok
```

Következő szavunk a >**BINARY** vagy más néven **CONVERT** (cim1 --- d cim2). Használata előtt a verem tetején el kell

helyeznünk az átalakitandó füzér cimét (cim1). A konverzió cim+1-től kezdődik, és az első, számként nem értelmezhető karakterig tart. Az átalakitott számot (d) a verem 2. szintjére, a maradék füzér cimét (cim2) pedig a verem tetejére helyezi. A fig-FORTH ezt az utasitást **(NUMBER)** néven ismeri.

FIGYELEM: mind a NUMBER, mind a CONVERT utasitás a BASE figyelembevételével végzi az átalakitást.

2.7.3. Konverzió karakterből számjeggyé

Ezt az átalakitást a **DIGIT** utasitás (n1 n2 --- n3 j vagy j) végzi. Az n1 számot egy karakter ASCII kódjaként értelmezi, és n2-t számrendszerként alapul véve megkisérli számmá, digitté konvertálni. Ha a karakter létező számjegye a számrendszernek, ezt a verem 2. szintjére, a verem tetejére pedig egy logikai igaz értéket (j=-1) tesz. Ellenkező esetben csupán egy hamis (j=0) értéket kapunk. Például az "3" karakter ASCII kódja 51, ezt alakitsuk számmá a tizes számrendszerben:

```
51 10 DIGIT . . -1 3 ok
```

Az "A" betű kódja 65:

```
65 10 DIGIT . 0 ok
```

De:

```
65 16 DIGIT . . -1 10 ok
```

2.8. Szövegek és karakterfüzér-műveletek

A FORTH - eltérően más magasszintű nyelvektől (elsősorban a BASIC-től) - közvetlenül nem teszi lehetővé a karakterfüzérekkel való műveletvégzést. Viszont rugalmasságából adódóan a programozó gyakorlatilag bármely más nyelvben megszokott karakterfüzér műveletet létrehozhat akár assembler, akár magasszintű nyelvi eszközök segitségével. A továbbiakban ismertetjük a FORTH-79 szabvány által tartalmazott szövegargumentumú utasitásokat.

A **-TRAILING** (cim hossz1 --- cim hossz2) bemenő adatként egy karakterfüzér cimét, és fölötte a karakterfüzér hosszát várja. A megcimzett szöveg végéről leválasztja az esetleg ott levő szóköz karaktereket. A vermen levő cim a helyén marad. A hosszat tartalmazó számot leveszi, a valós hossznak megfelelően korrigálja, és visszahelyezi a veremre.

A **COUNT** utasitás (cim --- cim1 hossz, cim1=cim+1) a verem tetején levő cimen található karakterfüzér hosszát helyezi a veremre. A cimet úgy módositja, hogy az első karakterre mutasson. Belátható, hogy a COUNT utasitás egyenértékű az alábbi utasitássorozattal:

```
COUNT = DUP C@ SWAP 1+ SWAP
```

A FORTH a beirt szövegeket egy bemeneti tárolóba (<u>input puffer</u>) olvassa be. Ez a memóriának általában az $0, vagy a TIB (Terminal Input Buffer) rendszerváltozó által meghatározott cimén kezdődik.

A **TEXT** szó (n ---) az input pufferből beolvas egy, a verem tetején megadott karakterrel lezárt karakterfüzért, majd ezt eltárolja a szövegpufferbe. Előzőleg ezt a memóriatartományt feltölti 64 szóköz (ASCII 32) karakterrel. A szövegpuffer első cimére kerül a füzér hossza, PAD+1-től kezdődően pedig a füzér karaktereinek ASCII kódjai. A lezáró

karakter nem eleme a füzérnek. Például:

```
65 TEXT BCDA ok          ( lezáró karakter: ASCII 65 = A )
PAD C@ . 3 ok            ( füzér hosszának kiiratása )
PAD 1+ C@ EMIT B ok      ( első karakter kiiratása )
```

A TEXT párja a **WORD** szó (n --- cim, vagy n ---). Az input pufferből beolvas egy, a verem tetején megadott karakterrel lezárt karakterfüzért, majd ezt eltárolja a későbbiekben részletezendő DP rendszerváltozó által mutatott területre. Előzőleg ezt a memóriatartományt feltölti 64 szóköz (ASCII 32) karakterrel. A első cimre kerül a füzér hossza, utána pedig a füzér karaktereinek ASCII kódjai. A lezáró karakter nem eleme a füzérnek. A FORTH-79 szerint az utasitás a verem tetejére teszi DP értékét (cim), a fig-FORTH-beli WORD azonban nem. Használata interpreter módban nem ad értelmes eredményt, igy példát majd a 3.7.9. feladat megoldásánál adunk.

Az **ENCLOSE** (cim n1 --- cim n2 n3 n4) egy szöveg-kereső szó. Bemenő paraméterként egy karakterfüzér kezdőcimét, és fölötte a szöveg lezáró karakterének kódját várja. A cimet a vermen hagyja, fölötte megadja az első nem határoló karakter sorszámát (n2), a szöveg utolsó, nem határoló karakterének sorszámát (n3), és a szöveg utáni első határoló karakter sorszámát (n4). Jegyezzük meg, hogy a nullkarakter (ASCII 0) határoló karakternek számit! Bemutatunk egy példát a jobb érthetőség érdekében. Tegyük fel, hogy PAD-től le van téve az AAABCDEFA karakterfüzér, ebből maga a szöveg BCDEF, lezáró karakter az A, kódja 65:

```
PAD 65 ENCLOSE ok
. 9 ok                   ( n4, az utolsó A betű sorszáma )
. 8 ok                   ( n3, az F betű sorszáma )
. 3 ok                   ( n2, a B betű sorszáma )
. 11651 ok               ( PAD cime, ettől eltérő lehet )
```

2.8. Szövegek és karakterfüzér-müveletek

A csoport utolsó tagja a -TEXT utasitás (cim1 n cim2 --- n=-1 vagy 0 vagy 1), karakterfüzérek összehasonlitására szolgál. Bemenő veremszintek: a 3. szinten az egyik karakterfüzér (füzér1) cime (cim1), a 2. szinten az összehasonlitandó karakterek száma (alkalmasint a rövidebb füzér hossza), a verem tetején a másik karakterfüzér (füzér2) cime (cim2). Az utasitás karakterről karakterre összehasonlitja a két karakterfüzért, és logikai hamis értékkel tér vissza, ha egyenlőek, egyébként igazzal. Ha füzér1 nagyobb füzér2-nél, akkor az igaz érték pozitiv, ha füzér1 kisebb füzér2-nél, akkor negativ. Füzérekre a kisebb relációt a lexikografikus rendezés szerint értelmezzük, azaz füzér1 kisebb füzér2-nél, ha füzér1 első eltérő karakterének ASCII kódja kisebb füzér2 megfelelő karakterének ASCII kódjánál.

Példánkban tegyük fel,hogy szövegpuffer elejére (PAD) le van téve a "bukfenc", a PAD+16 cimen pedig a "bukkanó" karakterfüzér:

```
PAD 7 PAD 16 + -TEXT . -1 ok
```

A fentiek alapján az Olvasó könnyen megértheti a példa működését.

2.9. Ki- és beviteli müveletek

A programozási nyelvekben a legfontosabb az ember-gép kapcsolat. Ezáltal a ki- és beviteli utasitások meghatározhatják egy nyelv használhatóságát.

2.9.1. Számok kiiratása

A FORTH nyelv számok kivitelére szolgáló utasitások széles skáláját bocsátja a felhasználó rendelkezésére,

meghagyva a lehetőséget különleges igények esetén saját kiirási formátum létrehozására (<# # #> utasitások, l. a 2.7.2.1. pontot). Az alábbi utasitások a BASE értékének megfelelő formában végzik a kiirást.

Legfontosabb ilyen utasitás a ., (n ---). Működése során az adatvermen levő 16 bites előjeles számot kiirja, egy szóközzel követve.

Az U. 16 bites előjel nélküli számok kiirására alkalmas (u ---). Pl.:

```
-1 . -1 ok
-1 U. 65535 ok
```

A .R (n m ---) előjeles egész számok m mezőszélességben történő kiirását végzi. Különösen előnyös táblázatok készitésénél. A számot jobbra tömöritve irja ki. Pl.:

```
1 3 .R  1 ok
1 6 .R     1 ok
```

Az U.R (u m ---) az előző utasitás megfelelője 16 bites előjel nélküli számokra.

A D. utasitás a . duplapontosságú megfelelője (d ---).

A D.R az U.R megfelelője duplapontos egész számokra (d m ---). Hivásakor a verem tetején kell átadni a mezőszélességet, és alatta a kiirandó kétszeres pontosságú számot. Például:

```
1234567.8 9 D.R 12345678 ok
1234567.8 12 D.R   12345678 ok
```

Változók, memóriacimek tartalmának kiirását végzi a ? (cim ---).

? = @ .

Például:

BASE ? 10 ok

(Ettől persze nem lettünk okosabbak, mert bármi is a BASE értéke, mindig 10-ként iródik ki, hiszen az alapszám éppen a BASE értéke.)

2.9.2. Karakterek és karakterfüzérek kivitele

Egy karakter kivitelére az **EMIT** utasitás használható (n ---). Bemenő adata az adatvermen levő 16 bites szám alsó 7 bitje. Ezt a FORTH ASCII kódként értelmezi és az annak megfelelő karaktert kiirja a képernyőre.

65 EMIT A ok

A **SPACE** utasitás (---) nem más, mint a 32 EMIT, ahol 32 a szóköz kódja.

Az n **SPACES** (n ---) a SPACE n-szeri végrehajtására utasitja a FORTH-ot (n darab szóköz kiirása).

A **CR** utasitás (---) egy soremelést eredményez. Bizonyos rendszerekben 13 EMIT, másokban 13 EMIT 10 EMIT utasitásokkal helyettesithető.

A képernyő letörlésére szolgál a **PAGE** utasitás (---).

Amennyiben egy karakterfüzérrel nem kivánunk müveletet végezni, csak kiiratni, ugy használjuk a ." karakterfüzér" formulát.Pl.:

." buta pelda" buta pelda ok

A **TYPE** utasitás (cim n ---) a memóriában található karakterfüzérek kivitelére alkalmas. Két adatot igényel: a karakterfüzér hosszát (n) és cimét (cim).

2.9.3. Karakter- és karakterfüzér-bevitel

A billentyűzetről vagy más beviteli eszközről a **KEY** utasitás (--- n) egy karaktert olvas, amelynek ASCII kódját (n) a veremre helyezi.

KEY .<R> 65 ok

(A továbbiakban, ahol ennek különös jelentősége van, <R>-rel jelöljük a RETURN gomb lenyomását.) A KEY egy billentyű leütésére várakozik. A gomb lenyomásakor annak ASCII kódja a veremre kerül, majd ezt a pont kiirja (65 az "A" betű ASCII kódja).

Az **EXPECT** parancs (cim n ---) végrehajtásakor a FORTH az adatverem tetején levő számnak (n) megfelelő számú karaktert olvas be a beviteli eszközről (általában a billentyűzetről). A verem második szintjére egy cimet kell elhelyeznünk , amely a füzér tárolására szánt memóriarészre mutat. A beolvasás RETURN (ASCII 13) hatására félbeszakad. A füzért 0 karakterrel zárja le.

Példánk a billentyűzetről beolvas 4 karaktert és ezek ASCII kódjait leteszi a szövegpufferbe (PAD):

```
PAD 4 EXPECT<R> ABCD ok   ( ABCD-t a klaviatúráról várja )
PAD C@ EMIT A ok          ( első karakter kiiratása )
```

A **QUERY** utasitás (---) a 2.8. szakaszban leirt input pufferbe beolvas egy legfeljebb 80 karakter hosszú, RETURN-nel lezárt karakterfüzért, majd a 6.7. szakaszban

részletezendő IN rendszerváltozót nullára állitja.

```
QUERY = TIB @ 80 EXPECT 0 IN !
```

Mivel a FORTH interpreter a TIB @ cimtől kezdődően az IN @ poziciójú szöveget olvassa be, a QUERY végrehajtása után a pufferbe beolvasott szöveg rögtön végre is hajtódik. Például vigyük be QUERY-vel az 1 . utasitássorozatot:

```
QUERY<R> 1 .<R> 1 ok
```

2.10. Feladatok

Az alábbi feladatok megoldásait az F1 függelékben adjuk meg.

2.10.1. Az 1 2 3 4 utasitássorozat beirása után irassuk ki a számokat a beirás sorrendjében, valahogy igy:

```
1 2 3 4 ( utasitások ) 1 2 3 4 ok
```

2.10.2. Valósitsa meg a 2PICK (duplapontos PICK) utasitást!

```
123456.6 789100. 3459.2 3 ( utasitások ) ok
D. D. D. D. 1234566 34592 789100 1234566 ok
```

2.10.3. Valósitsa meg a 2ROLL (duplapontos ROLL) utasitást!

```
123456.6 789100. 3459.2 3 ( utasitások ) ok
D. D. D. 1234566 34592 789100 ok
```

2.10.4. A PAD utasitás által megadott cimen található 16 bites adatot cimként értelmezve helyezze az e cim által mutatott memóriarekesz 8 bites tartalmát a veremre!

2.10.5. Olvasson be két karaktert a billentyűzetről! A kisebb ASCII kódú karaktert irassa ki a képernyőre!

2.10.6. Olvasson be egy 10 számjegyet tartalmazó karakterfüzért a billentyűzetről! Alakitsa át 32 bites előjeles számmá, és irassa ki az alábbi formában:

x,xxx,xxx,xxx

ahol x egy-egy számjegyet jelöl.

3. A FORTH mint compiler

Könyvünk előző fejezeteiben megismertük azt, hogy miként működik a FORTH interpreter módban. Eddig a billentyűzetről bevitt utasitássorokat a FORTH azonnal feldolgozta és végrehajtotta. Ez a működési mód megegyezik a különféle BASIC rendszerek képernyő üzemmódjával.

Igy a FORTH mint intelligens kalkulátor működött, de programot irni még nem tudtunk. Kérdés tehát, hogy miképpen lehet az ismétlődő vagy logikailag összefüggő tevékenységeket csoportositani és egy különálló egységként kezelni.

Lássunk erre egy példát:

```
DUP * SWAP DUP * +
```

Példánk a verem tetején levő két szám négyzetének összegét állitja elő.

Láthatjuk, hogy a " DUP * " utasitássorozat kétszer fordul elő, igy jó lenne ezt egy egységként kezelni. Emellett "nevén is kellene nevezni a gyereket" úgy, hogy a névből funkciójára is következtethessünk. (Például a neve lehetne NEGYZET.)

Hogyan oldhatjuk ezt meg FORTH-ban? A választ a következő pontok adják meg.

3.1. Algoritmusabsztrakció

Algoritmusnak nevezzük az azonos tipusú feladatok megoldási szabályainak lépésekbe rendezését. Előző példánknál

maradva a " DUP * " is egy (igaz, igen egyszerű) algoritmus.

A számitástechnikában általában kétféle algoritmustipust különböztetünk meg:

- Konkrét algoritmusok, amelyek szerves részei az adott programozási nyelvnek. Ezek minden előkészitités nélkül a programozó rendelkezésére állnak, és az algoritmus megvalósitásáról a fordítóprogram készitői gondoskodtak. Az eddig megismert összes művelet, utasitás (például a +, -, <, >, ROT, SWAP és még sorolhatnánk tovább) egy-egy konkrét algoritmus.

- Absztrakt algoritmusok, amelyeket a programozó hoz létre az adott nyelv szintaktikájának megfelelően. Az algoritmus felépitéséhez a program irója felhasználhatja egyrészt a rendelkezésére álló konkrét, másrészt az általa előzőleg létrehozott absztrakt algoritmusokat is.

Az absztrakt algoritmusok létrehozásának célja kettős.

1. Egyrészt az adott tevékenységet felépitő részleteket mintegy elrejtjük az algoritmusba.

belépő paraméter --> ALGORITMUS --> kilépő paraméter

A kipróbált és működő tevékenységet már tekinthetjük fekete doboznak is, azaz számunkra érdektelen, hogy a

kilépő paraméter=f(belépő paraméter)

függvény hogyan valósul meg.

2. Másrészt az absztrakció során az algoritmusnak nevet is adunk, amely utalhat annak funkciójára és emellett a program szövegét olvashatóbbá és áttekinthetőbbé teheti.

Gondosan ügyeljünk a világos, kifejező névválasztásra! Tapasztalatok szerint a jól megválasztott azonositókkal a program javitásakor, bővitésekor jelentős időt és fáradságot takarithatunk meg.

Ezek után nézzük meg, milyen absztrakciós eszközzel rendelkezik a FORTH nyelv! Ezt az eszközt, amely egyben a nyelv legalapvetőbb egysége, szónak nevezzük.

3.1.1. A szó mint a FORTH program alapegysége

Minden programozási nyelvben találhatunk olyan szintaktikai egységeket, amelyek működés szempontjából tovább már nem bonthatók. Lássunk néhány példát erre:

- LD A,0F9H. Ez egy Z80 assembly sor, amely az A regiszterbe hexadecimális F9-et tölt. Működés szempontjából ezt kisebb egységre nem bonthatjuk, hisz bármely elem (LD, A, 0F9H) elhagyása a sort értelmetlenné tenné.

- VAR1:=VAR2; Ez egy PASCAL értékadás. Erre is igazak a fenti megállapitások, azaz egyik elemet sem hagyhatjuk el. Ugyanigy találhatunk más olyan egységeket, amelyek a végrehajtás szempontjából egynek tekintendők.

A FORTH nyelvben ez a működési alapegység, mint már emlitettük, a szó. Minden eddig megismert művelet, utasitás egy szó (DUP, DROP, VARIABLE, <, >, + stb.). Minden szóhoz tartozik egy adott tevékenységsorozat, amelyet a FORTH rendszernek a szó felismerésekor végre kell hajtania.

Nézzük meg például, milyen tevékenységeket hajt végre egy olyan, látszólag primitiv szó, mint a "<" logikai művelet:

1. Megvizsgálja, hogy a verem második szintjén levő szám kisebb-e a verem tetején levőnél.

2. Megjegyzi az összehasonlitás eredményét, és a két operandust leveszi a veremről.

3. Ha a a művelet eredménye igaz volt, akkor minusz egyet, ha hamis, akkor nullát tesz a verem tetejére.

Azokat a tevékenységeket, amelyeket az adott szó reprezentál, a FORTH rendszer egy úgynevezett **szótár**ban tárol. Ennek felépitése teljesen hasonló például a Magyar Nyelv értelmező Szótárához. Lássunk utóbbiból egy cimszót:

program számitógép műveleteinek sorrendjét meghatárczó utasitások sorozata.

⟨-----⟩ ⟨----------------------------⟩

cimszó meghatározás

Ehhez hasonlóan tárolja saját szótárában a FORTH a szavak jelentését. A "⟨" műveletnél tehát a "⟨" a cimszó, az 1., 2., 3. tevékenységek pedig a cimszóhoz tartozó meghatározást alkotják. Minden szó ezen rendszer szerint tárolódik. A szótárról részletesen a 3.2.1. pontban és a 6.5. szakaszban szólunk.

Az eddig megismert szavakhoz tartozó algoritmusok elkészitéséről és azoknak a szótárban való elhelyezéséről a FORTH rendszer elkészitői gondoskodtak. Ezeket a szavakat a rendszer inditása után a felhasználó minden előkészités és definició nélkül használhatja.

Emellett a programozó maga is bővitheti a szótárat úgy, hogy algoritmusabsztrakcióval új szavakat hoz létre. Ezeket a FORTH a szótárban elhelyezi, és biztositja használatukat. A

szó definiciója négy jól elkülönithető szintaktikai egységre bontható.

: név törzs ;

1. A **:** szó a szódefinició bevezető eleme, amely utasitja a FORTH rendszert, hogy interpreter módból compiler (szótárkészitő) módba kapcsoljon.

2. A **név** az új szó által jelölt algoritmus azonositója, a következő karaktercsoportok elemeit tartalmazhatja tetszőleges kombinációban:

a) nagybetűk (A,B,C,...Z)
b) kisbetűk (a,b,c,...z)
c) számjegyek (1,2,3,...0)
d) speciális karakterek:

! " # $ % & ' ( ) * + ,
- . / : ; < = > ? @ [ ] ^

Az azonositók legfeljebb 31 karakterből állhatnak. Az új szó nevét a FORTH elhelyezi a szótárába. Ez lesz a szódefinició cimszava.

3. A **törzs** azokat a tevékenységeket tartalmazza, amelyeket az új szó felismerése után végre kell hajtani. Ez a szófelsorolás tartalmazhat egyrészt alapszavakat (konkrét algoritmusokat), másrészt a felhasználó által már előzőleg definiált szavakat (absztrakt algoritmusokat).

A FORTH nyelv szigorúan predefinit, azaz a szó törzsében soha nem szerepelhet olyan szó, amit még nem definiáltunk, sőt az éppen definiálandó szó nevét sem tartalmazhatja. Ilyenformán tehát rekurzióra sincs

közvetlenül lehetőségünk. A rekurziót más eszközzel biztositja a nyelv, amelyről a 3.3.3. pontban részletesen szólunk.

4. A ; szó a szódefinició lezárását jelenti (tehát nem a RETURN billentyű megnyomása!). A szódefinició belsejében a kocsi-vissza csak elválasztókarakterként funkcionál és igy egyenértékű a szóközzel. Ezt felhasználhatjuk szavaink struktúrálására. Compiler módban a FORTH a szavakat nem végrehajtja, hanem a törzset alkotó szavak cimét az új szó meghatározás részében elhelyezi. Ha egy szó neve nem szerepel a szótárban, akkor a fordítás hibajelzéssel leáll (ismeretlen szó).

Ha a szódefinició hiba nélkül fejeződik be, egyik vermet sem rontja el, hiba esetén viszont mind az adat-, mind a visszatérési verem törlődik.

Lássunk példát egy új szó definiciójára:

```
: NEGYZET ( n --- n*n )
      DUP * ;
```

Ha visszalapozunk a fejezet elejére, ott éppen ezt a példát emlitettük. Az új szó neve NEGYZET, törzse pedig a végrehajtandó (jelen esetben két) szó, a DUP és a *. A zárójelek közötti rész megjegyzés, azaz a FORTH ezt nem veszi figyelembe, akár el is hagyható. A megjegyzés kiválóan felhasználható a verem állapotainak jelzésére. Ezzel a szó működése könnyebben megérthető. A szó definiálása után a NEGYZET-et már úgy használhatjuk, mintha mindig is a FORTH része lett volna. Mivel a szódefinició után interpreter állapotba kerültünk, szavunkat rögtön ki is próbálhatjuk.

```
5 NEGYZET . 25 ok
```

Először a FORTH a veremre helyezi az ötöt, majd végrehajtja a NEGYZET szót, amelynek hatására a veremre az 5 négyzete, azaz 25 kerül. Ezt a . szóval kiiratjuk. A hibamentes végrehajtást a FORTH az "ok" üzenettel jelzi.

Itt szeretnénk kitérni arra, hogy bár a 4. fejezet részletesen foglalkozik a helyes FORTH programozási stilussal, két dolgot már most jegyezzünk meg:

- Minden új szavunkat a definició után rögtön próbáljunk ki úgy, ahogy az előbb láttuk. Leghelyesebb, ha legalább két-három különböző bemenő adatra vagy adatcsoportra végezzük el a kipróbálást. Irassuk ki az eredményeket és vizsgáljuk meg algoritmusunk helyességét! Csak akkor lépjünk tovább, ha szavunkat kipróbáltuk és jónak találtuk!

- Törekedjünk arra, hogy a szó törzse lehetőleg kevés szót tartalmazzon! Minél több és rövidebb szóból áll programunk, annál áttekinthetőbb és javithatóbb lesz.

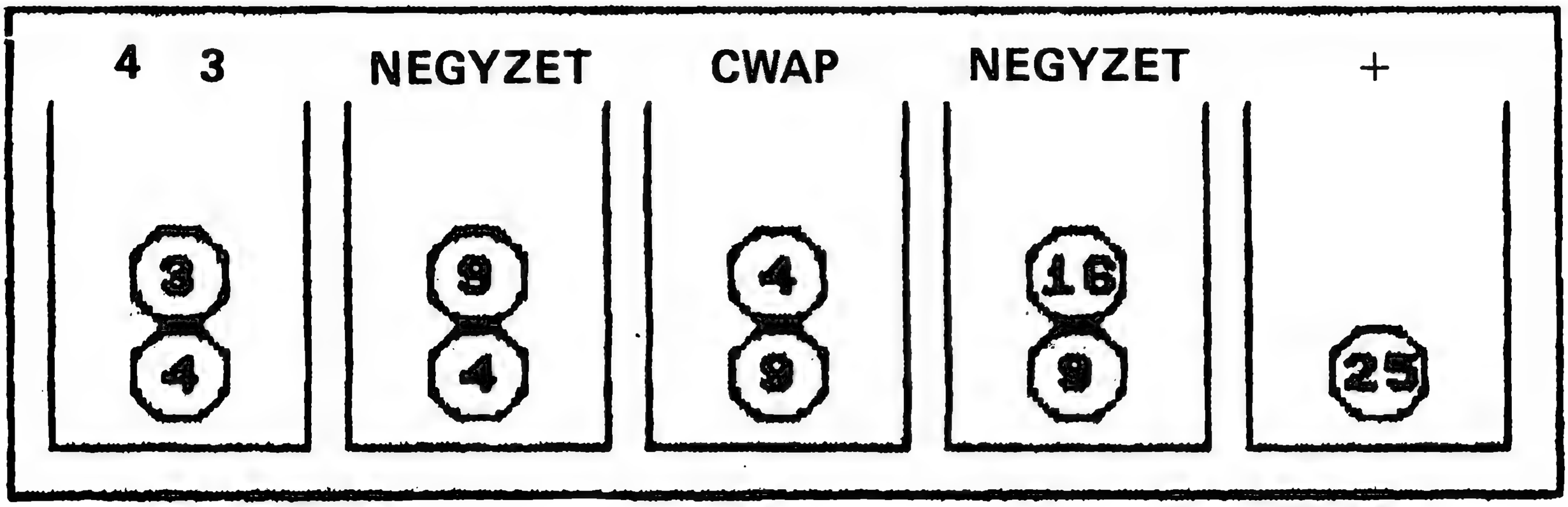
Nézzük, hogyan alakul ezután a fejezet elején megkezdett példánk! Továbbra is a vermen levő két szám négyzetösszegét szeretnénk előállitani, de most már a NEGYZET szót is felhasználva. A parancssor most igy alakul:

```
NEGYZET SWAP NEGYZET +
```

Ezt a sort begépelve konkrét értékekkel, a következőt kapjuk:

```
4 3 NEGYZET SWAP NEGYZET + . 25 ok
```

A sor működését könnyen megérthetjük a 3.1. ábrából.



3.1. ábra
A verem állapotai a négyzetösszeg számitása közben

Most azt szeretnénk, hogy ne kelljen a fenti parancssort mindig leirnunk, ha két szám négyzetösszegét akarjuk kiszámittatni. Ehhez definiáljuk az A*A+B*B szót, ami ezt meg fogja tenni. Természetesen ehhez a NEGYZET szót is felhasználjuk.

```
: A*A+B*B           ( n1 n2 - n2*n2+n1*n1 )
          NEGYZET   ( n1 n2*n2 )
          SWAP      ( n2*n2 n1 )
          NEGYZET   ( n2*n2 n1*n1 )
          +         ( n2*n2+nn1*n1 ) ;
```

Vegyük észre, hogy nem tettünk mást, mint az eddig használt parancssort egyszerűen beirtuk egy szó törzsébe. Próbáljuk is ki új szavunkat:

```
4 3 A*A+B*B . 25 ok
```

Láthatjuk, hogy a szódefinició lehetőségével a FORTH igen hatásos eszközt ad a kezünkbe. A következő pontokban a szavak és a szótár további tulajdonságaival ismerkedünk meg.

3.1.2. A szó mint eljárás

A FORTH szavak a fordítás és a futtatás alatt eljárásként viselkednek, hasonlóan a BASIC, FORTRAN és assembly szubrutinokhoz vagy a PASCAL és ALGOL procedure-höz.

Interpreter módban a FORTH a szó megtalálása után a "meghatározás" rész szavait végrehajtja.

Compiler üzemmódban az új szó "meghatározás" részébe bekerülnek a törzset alkotó szavak cimei, és a FORTH a végrehajtáskor fejti ki azokat.

Mivel tehát a fordítás során nem történik behelyettesitès (hiszen a szó nem makróként viselkedik), a program bonyolultságának növekedése soha nem vezet jelentős méretnövekedéshez. Ez is alátámasztja azt az előző megállapitást, miszerint a programot ajánlatos több rövid szóból felépiteni.

3.1.3. A veremszintek mint formális paraméterek

A FORTH nem biztosit lehetőséget formális paraméterek használatára és futásidejű formális-aktuális paramétercserére úgy, mint más magasszintű (PASCAL, ALGOL, FORTRAN stb.) nyelvek. A FORTH veremgépként való viselkedése miatt viszont erre nincs is szükség.

3.1. Algoritmusabsztrakció

A FORTH-ban a formális paraméterek szerepét a veremszintek veszik át. Egy szó definiciója előtt mindig eldöntjük, hogy a bemenő paraméterek a veremben hol és milyen sorrendben helyezkednek el. Ennek rögzitése után úgy irjuk meg szavunkat, hogy az az egyes paramétereket az általunk meghatározott veremszinteken várja. A szó definiciója előtt természetesen azt is eldöntjük, hogy a kimenő paramétereket a szó a verembe milyen sorrendben rakja le.

A 3.1.1. pontban definiált A*A+B*B szavunk esetében például feltételeztük, hogy a két bemenő paraméter a verem tetején, egymás alatt helyezkedik el. Azt is tudjuk, hogy szavunk a két számot a veremről leveszi és a verem tetejére eredményként négyzetösszegüket helyezi.

Egy másik lehetőség az, hogy a szó bemenő paraméterét vagy paramétereit változókban helyezzük el és a kimenő paraméterek is változókba kerülnek. Ekkor a szó meghivása előtt gondoskodnunk kell a megfelelő változók értékének beállitásáról.

Irjuk át A*A+B*B szavunkat olyanra, hogy az két bemenő paraméterét a PARAM.1 és PARAM.2 változókban várja, és az eredményt az EREDMENY változóba teszi. Ehhez definiálnunk kell a három változót, valamint gondoskodnunk kell arról, hogy azok ertéke a veremre kerüljön:

```
VARIABLE PARAM.1
VARIABLE PARAM.2
VARIABLE EREDMENY
: A*A+B*B PARAM.1 @ NEGYZET
          PARAM.2 @ NEGYZET +
          EREDMENY ! ;
```

Ezután szavunk igy működik:

```
4 PARAM.1 ! 3 PARAM.2 ! ok
A*A+B*B EREDMENY ? 25 ok
```

Természetesen a két módszert (paraméter a vermen - paraméter változóban) keverhetjük is. A szó például változókban várja bemeneti paramétereit, de az eredményt (vagy eredményeket) a veremre teszi stb.

Mindazonáltal törekedjünk arra, hogy szavaink lehetőleg a veremszinteket használják paraméterként. Egyrészt ez rövidebb kódot és gyorsabb működést eredményez, másrészt jobban illeszkedik a FORTH filozófiájához is (minthogy a legtöbb FORTH alapszó a vermet használja paraméterezésre).

3.2. A szavak használata

A következőkben kicsit részletesebben megismerkedünk a szótárral, néhány rendszervezérlő szóval és azzal, hogy miképp befolyásolhatjuk a FORTH működését compiler üzemmódban. Emellett a szavak néhány új tulajdonságát is bemutatjuk.

3.2.1. A szótár felépitése

3.2.1.1. Általános ismertetés

Azt már tudjuk, hogy a FORTH-ban a szótár a szavak tárolására szolgál. Szerepe kettős. Egyrészt compiler módban az a funkciója, hogy a FORTH megkeresse a szódefiníció törzsében levő szavakat, valamint meghatározza címüket. Másrészt interpreter módban a FORTH a szótárban levő szó

"meghatározás" részére adja a vezérlést, ami a szó végrehajtását jelenti.

Egy szó cimén azt a cimet értjük, ahol a szótári bejegyzés kezdődik az adott szóról. Emellett a szóhoz további cimeket is rendelhetünk. Ezeket jelenleg figyelmen kivül hagyhatjuk, használatukra csak bonyolultabb feladatok esetén van szükség. A 6. fejezet fogja ezt mélyrehatóan tárgyalni. Most csak annyi ismeretet közlünk, amennyi könyvünk további részeinek megértéséhez elengedhetetlenül szükséges.

A szó a szótárban két funkcionálisan különálló részre osztható:

- A **szó feje** tartalmazza a szó nevét (cimszó) ASCII karakterek formájában. Emellett más, a FORTH működéséhez szükséges információkat is magába foglal. A fej arra szolgál, hogy a FORTH azonositani tudja az adott szót. A szó cimét is a fej információi alapján állitja elő.

- A **szó meghatározás része.** Ez tartalmazhat egyrészt cimlistát, másrészt gépi kódú rutinokat. A gépi kódú részeket a FORTH rendszer készitői irták meg, és a legalapvetőbb FORTH rutinokat valósitják meg.

A szótár maga két részből áll:

1. Alap- vagy beépitett szótár. Ez tartalmazza a konkrét algoritmusokat. Az alapszótár is további két részre bontható:

a) gépi kódban irt szavak. Ezek képezik a FORTH alapját.

b) FORTH-ban irt szavak. Ezek mind gépi kódban, mind FORTH-ban megirt szavakat felhasználhatnak.

2. Felhasználó által definiált szótár. Ezen szótár szavai általában 1b) tipusúak (bár a beépitett ASSEMBLER segitségével 1a) tipusú szavak is létrehozhatók, de erről majd később lesz szó).

Az alap- és a felhasználói szótár az esetek többségében teljesen egyenértékű a FORTH szempontjából. Néhány esetben mégis a FORTH a két szótárt eltérően értelmezi. Ezt a megfelelő helyen mindig jelezni fogjuk.

3.2.1.2. Szótárkezelő utasitások

Az Olvasóban talán már felmerült az a kérdés, hogy mi a teendő akkor, ha meg szeretné nézni eddig definiált szavait, esetleg egy rossz vagy már szükségtelen szót ki szeretne törölni a szótárból. Lássuk, miképpen biztosit lehetőséget ezekre a FORTH.

A szótárban levő szavak kilistázásához használjuk a **VLIST** utasitást. Hatására a FORTH kiirja mind az alap-, mind a felhasználói szótárban levő szavak nevét a szavak definiciójához képest forditott sorrendben (az utolsóként definiált szóval kezdve). Azt, hogy a szavak törzse mi módon listázható ki, csak az 5.6. szakaszban fogjuk látni, ehhez ugyanis még további ismeretekre van szükség.

A szótárból törölhetünk is. Erre általában akkor van szükség, ha egy szavunk hibás, azaz rossz algoritmust valósitottunk meg. Az utasitás szintaktikája a következő:

FORGET szónév

A FORGET (= felejtsd el) utasitás törli a szótárból a

"szónév" nevű és az összes utána következő szót. "szónév" lehet változó- és konstansnév is. Térjünk vissza jól ismert négyzetösszeg-számoló példánkhoz! Először a NEGYZET szót definiáltuk:

```
: NEGYZET DUP * ;
```

Ezután jön az A*A+B*B szó:

```
: A*A+B*B NEGYZET NEGYZET + ;
```

Ha szavunkat kipróbáljuk, a következőt kapjuk:

```
4 3 A*A+B*B . 85 ok
```

Láthatjuk, hogy valami baj történt a szóval, hiszen 4*4+3*3 nem 85. Megvizsgálva a szót észrevehetjük, hogy a két NEGYZET utasitás közé elfelejtettük odairni a SWAP-et. Kitöröljük tehát a szótárból a rossz A*A+B*B-t a FORGET segitségével, és újra, most már helyesen beirjuk. Ezt a következő utasitásokkal érjük el:

```
FORGET A*A+B*B
: A*A+B*B NEGYZET SWAP NEGYZET + ;
```

Jegyezzük meg, hogy a FORGET NEGYZET utasitás hatására mind a NEGYZET, mind az A*A+B*B szavak törlődnek a szótárból.

Munkánk során előfordulhat, hogy a szótár eddig elkészült részét meg szeretnénk védeni a véletlen kitörléstől. A FORTH erre is lehetőséget ad. Ennek bemutatása előtt meg kell ismerkednünk két további szóval.

A FORTH a szótár következő üres helyének cimét a DP (Dictionary Pointer = szótármutató) nevű rendszer-

változóban tárolja. A változó a DP @ utasitássorozattal kiolvasható. Igy mindig megtudhatjuk azt, hogy a FORTH hol tart a szótár betöltésével. Annak érdekében, hogy ezt a kitüntetett szerepű változót ne kelljen mindig a "DP @" utasitásokkal kiolvasni, a FORTH a szótármutató értékének veremre helyezésére külön szót biztosit. Ez a **HERE** (--- n) utasitás, jelentése: itt.

DP @ = HERE

Az utasitás segitségével például megállapithatjuk, hogy a NEGYZET szó hány bájtot foglal el a szótárban.

```
HERE . 2289 ok
: NEGYZET DUP * ; ok
HERE . 2307 ok
2307 2289 - . 18 ok
```

Az első sorban kiirattuk a szótármutató értékét. Ez 2289 volt. A következő sorban definiáltuk a NEGYZET szót, majd a harmadik sorban ismét kiirattuk a DP értékét. A szódefinició után ez 2307 lett. A negyedik sorban a két cimet kivontuk egymásból és eredményül 18-at kaptunk. Ez azt jelenti, hogy a NEGYZET szó a szótárban 18 bájt helyet foglal el.

Láttuk tehát, hogyan férhetünk hozzá a szótármutató (DP) aktuális értékéhez. Ezt, valamint a FORGET szó egy eddig nem emlitett tulajdonságát fogjuk felhasználni szavaink véletlen törlés elleni védelmére.

A FORGET ugyanis úgy működik, hogy egy szót (és persze az azt követőeket) csak akkor törli a szótárból, ha annak cime nagyobb annál a cimnél, amit a FORTH a **FENCE** (= keritès) nevű változójában tárol. A FENCE a rendszer inditásakor az alapszótár végére, tehát a majdani

felhasználói szótár elejére mutat. Ily módon - feltéve, hogy a FENCE értékét nem változtattuk meg - akár az egész felhasználói szótárat kitörölhetjük a FORGET-tel. (Ez akkor történik, ha az általunk definiált szótár első szavát "felejtetjük el"). De nézzük meg, mi történik, ha a következő módon járunk el:

```
: SZO.1 ... ;
: SZO.2 ... ;
HERE FENCE !
: SZO.3 ... ;
: SZO.4 ... ;
```

Az első két sorban két szódefiníció szerepel. Tegyük fel, hogy a SZO.1 és SZO.2 szavakat kipróbáltuk, jónak találtuk és ezek után meg szeretnénk védeni a véletlen kitörléstől. Erre szolgál a harmadik sor. A HERE a veremre teszi a szótármutató (DP) értékét, amelyet ezután beirunk a FENCE változóba ("FENCE !"). Igy a SZO.1 és SZO.2 szavakra már hiába adjuk ki a FORGET utasitást, az hatástalan marad, hiszen cimeik kisebbek a FENCE-ben tárolt cimnél. A FENCE beállitása után definiált két szavunk, a SZO.3 és a SZO.4 persze bármikor kitörölhető. Ha ezeket is meg akarjuk védeni, csak a harmadik sort kell definiciójuk után megismételni.

Ha a védelem lehetőségével programunk során többször is élni szeretnénk, úgy érdemes egy VEDELEM nevű szót irni, amely előző példánk harmadik sorát tartalmazza.

```
: VEDELEM HERE FENCE ! ;
```

Ezek után bármely szódefiníció után leirjuk a VEDELEM szót, az az összes előzőleg megirt szavunkat törlésvédetté teszi. Természetesen ha a védelmet fel szeretnénk oldani, a FENCE-t eredeti értékére kell visszaállitanunk. Ezért érdemes rendszerinditáskor a FENCE-ben tárolt cimet megjegyezni.

Ehhez vezessük be a JEL nevű változót. Ebben fogjuk tárolni a FENCE kezdőértékét. Definiálni fogjuk továbbá a VISSZA szót is. Ez fogja visszaállitani a JEL felhasználásával FENCE eredeti értékét. Lássuk a szükséges definiciókat:

```
VARIABLE JEL
FENCE @ JEL !
: VISSZA JEL @ FENCE ! ;
```

A "VARIABLE JEL" deklarációval létrehozzuk a JEL változót, majd a következő sorban elmentjük a JEL-be a FENCE értékét. A harmadik sor a VISSZA definiciója. Működése igen egyszerű, a JEL-ban tárolt cimet a FENCE-be tölti. Nézzük meg, hogyan használhatjuk a VISSZA-t és a fentebb definiált VEDELEM szót:

```
( JEL, VISSZA, VEDELEM definiciói )
: SZO.1 ... ;
: SZO.2 ... ;
: SZO.3 ... ;
VEDELEM
: SZO.4 ... ;
: SZO.5 ... ;
VISSZA
FORGET SZO.3
VEDELEM
: SZO.3 ... ;
...
```

A programrészlet működését könnyen megérthetjük. Az első három szó definiciója után a VEDELEM-mel ezeket

megvédjük a kitörléstől. Ezek után újabb két szót hozunk létre, de ekkor rájövünk, hogy elképzelésünk hibás volt és SZO.3-tól kezdve szavainkat újra kell definiálni. (Ha programunkat nem megfelelő körültekintéssel tervezzük meg, a fenti eset igen gyakran előfordul).

Ahhoz, hogy a védett SZO.3-at kitörölhessük, előbb a VISSZA segitségével visszaállitjuk a FENCE-t a felhasználói szótár elejére. Ezután a "FORGET SZO.3" utasitással "elfelejtetjük" a SZO.3...SZO.5 szavakat, majd a megmaradt SZO.1 és SZO.2-t újból levédjük a VEDELEM utasitással. Ezek után hozzákezdhetünk a SZO.3 új definiciójának kidolgozásához.

3.2.2. Az újradefiniálhatóság és hatása

Ebben a pontban a FORTH egy igen érdekes és jól kihasználható tulajdonságáról szólunk. Nevezetesen arról, hogy lehetőségünk van a szótár szavait újradefiniálni, azaz egy szónévhez két (sőt több) meghatározást rendelni. Hasonló ez egy igazi értelmezőszótárhoz, ahol egy cimszóhoz több jelentés, meghatározás is tartozhat. Gondoljunk például az ár szóra, amely egyaránt jelenthet nagy viztömeget, cipészszerszámot és ellenértéket is. Azt, hogy a sok jelentés közül amelyiket használjuk, a szó környezete általában egyértelműen meghatározza.

A FORTH ennél sokkal egyszerűbben dönt arról, hogy éppen amelyik jelentést kell figyelembe vennie. Egy adott szó végrehajtása mindig a legutolsó definició törzsének végrehajtását jelenti. Ne felejtsük el viszont, hogy a szó újbóli definiálása nem jelenti az előző definició (vagy definiciók) törlését! Ezek a szótárban továbbra is megmaradnak, csak éppen új szavakban nem használhatók.

Mindezek után kövessünk végig egy olyan példasorozatot, amely remélhetőleg pontosan megmutatja az újradefiniálhatóság

szerepét és hatásait! A példa érzékelteti azt is, hogy általában mikor szoktuk a FORTH ezen tulajdonságát kihasználni: általunk létrehozott adatstruktúrákhoz a már megszokott jelölésrendszert (például műveleti jeleket) szeretnénk alkalmazni.

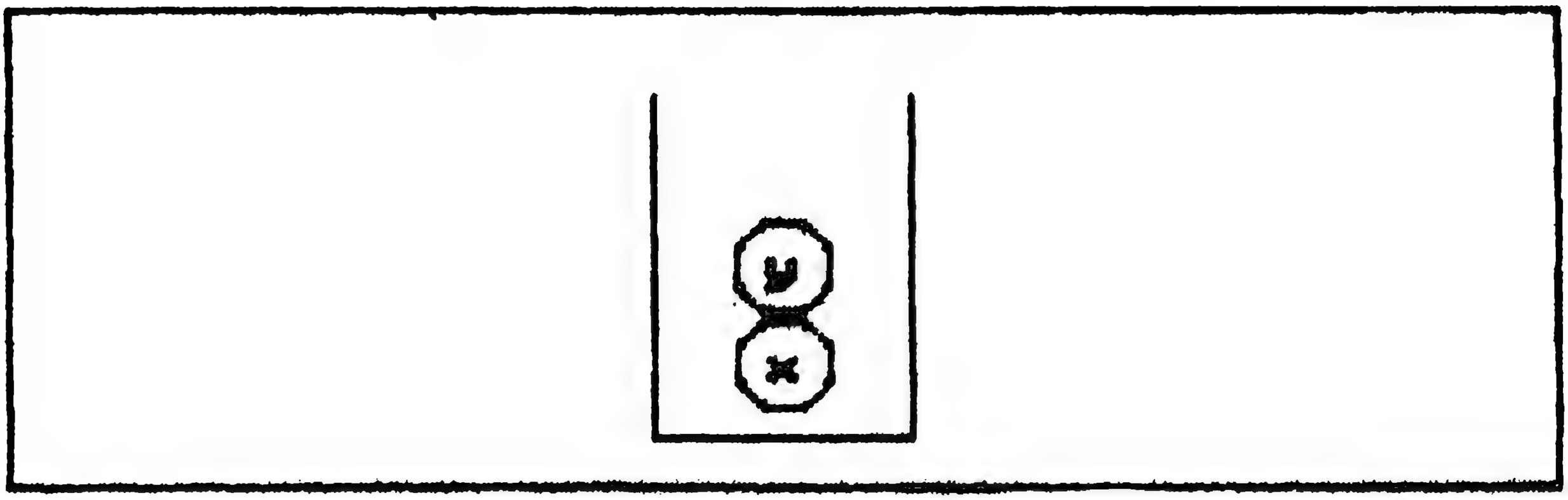
Ismert az, hogy a

$$z=x+i*y$$

alakú számokat komplex számoknak nevezzük, ahol x és y tetszőleges valós számok, i pedig a képzetes (imaginárius) egység, és i*i=-1. Az x-et z valós, y-t z képzetes részének nevezik, jelölésük:

$$x=Re(z) \qquad y=Im(z).$$

Tegyük fel, hogy számitásaink során szükség van arra, hogy komplex (és csak komplex) számokkal dolgozzunk. Előbb persze rögzitenünk kell azt, hogy miként fogjuk komplex számainkat a vermen ábrázolni. Tároljuk például a számokat valós rész, képzetes rész sorrendben! Igy tehát a z komplex szám a 3.2. ábrán látható módon helyezkedik el a vermen.



3.2. ábra
Komplex szám a vermen

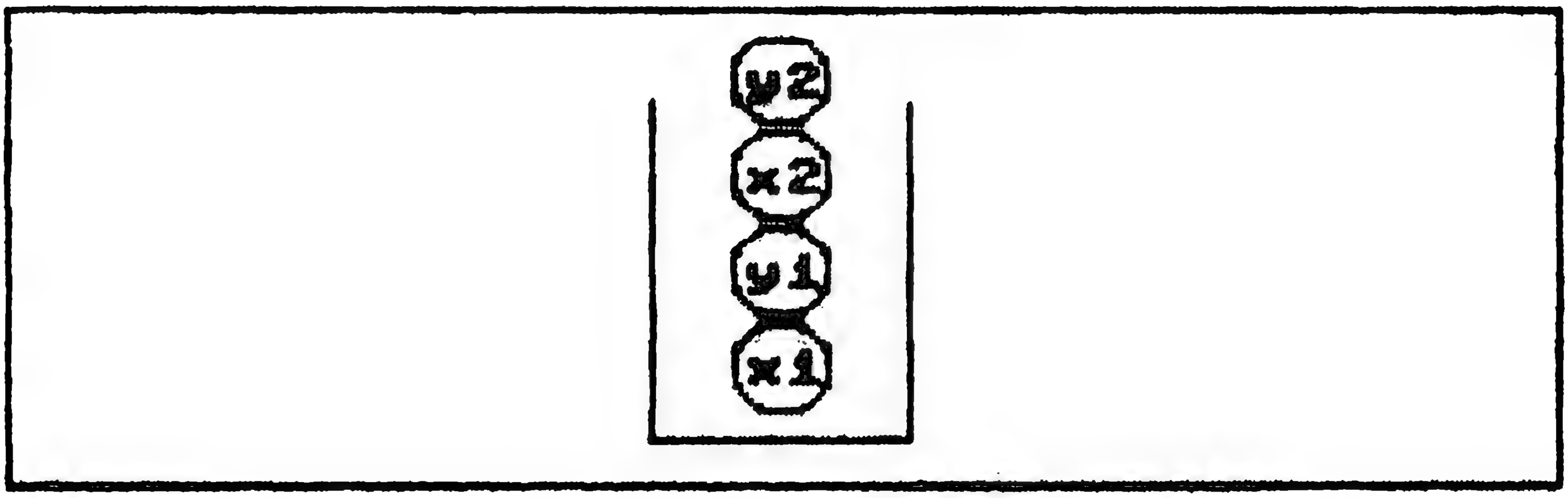
Ha a vermen a 3.2. ábrának megfelelően vannak a komplex számok, akkor szeretnénk, ha kiiratásukkor is a valós rész íródna ki először. Ezért definiáljuk a K. szót, ami ezt a kiírást végzi. A továbbiakban ezt fogjuk használni.

```
: K. SWAP . . ;
```

A komplex számokat egyelőre csak összeadni és kivonni szeretnénk. Persze szorozhatnánk és oszthatnánk is, de ezek képlete kicsit bonyolultabb, másrészt példánkban nincs is jelentősége. Lássuk a komplex összeadás és kivonás szabályait:

$$z_1+z_2=(x_1+x_2)+i*(y_1+y_2)$$
$$z_1-z_2=(x_1-x_2)+i*(y_1-y_2)$$

Foglalkozzunk előbb csak az összeadással. A művelet előtt a vermen elhelyezzük a z_1 és z_2 számokat (alul a z_1, fölötte a z_2). A verem állapota a 3.3. ábrán látható.



3.3. ábra
A z_1 és z_2 komplex számok a vermen

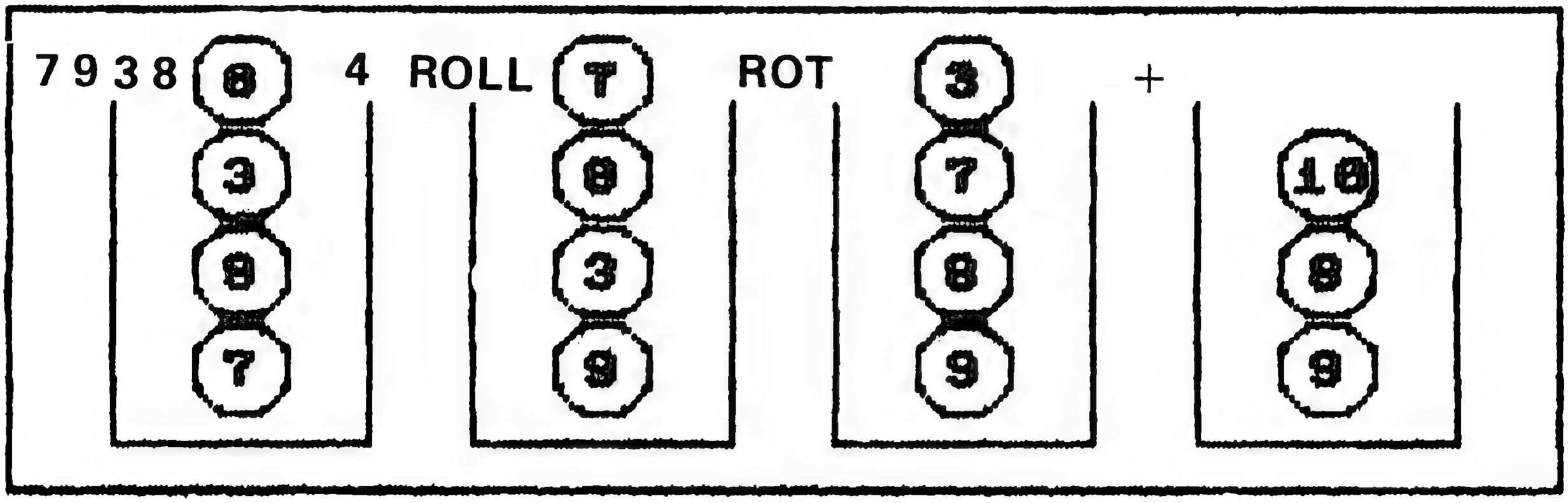
Irjuk meg most a K+ szót, amely z_1-et és z_2-t a már említett szabály szerint összeadja.

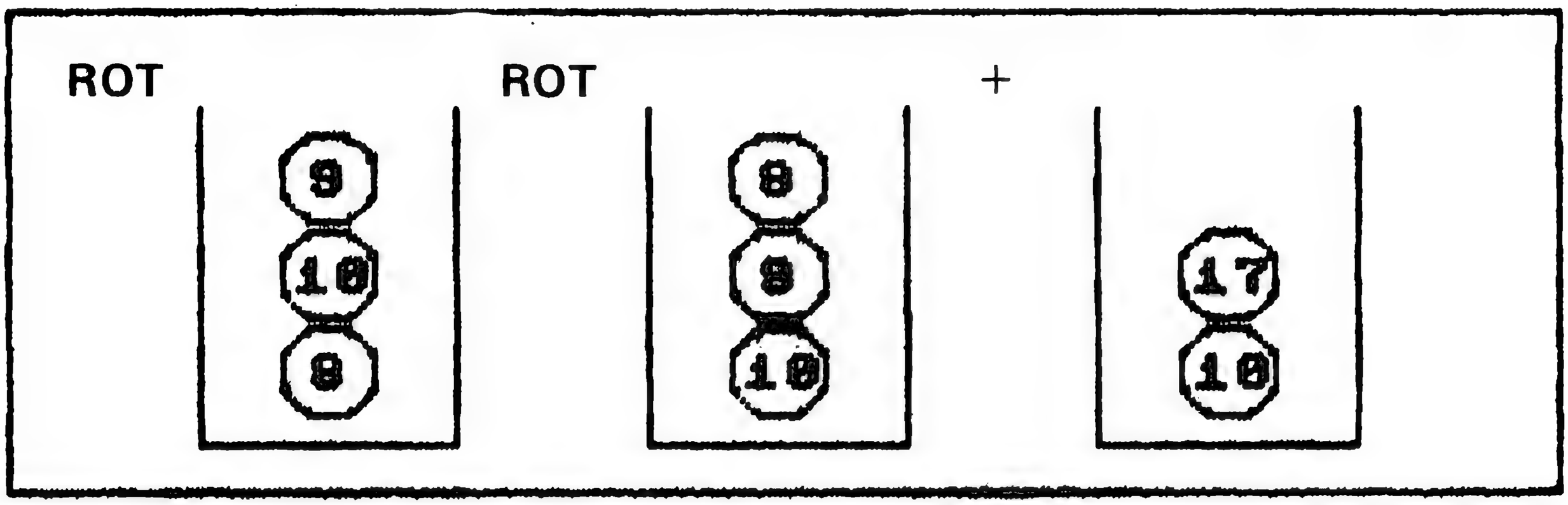
```
: K+ 4 ROLL ROT + ROT ROT + ;
```

Szavunkat ki is próbálhatjuk:

```
7 9 3 8 K+ K. 10 17 ok
```

Az 3.4. ábrán tanulmányozhatjuk a szó működését.





3.4. ábra
A verem állapotai a K+ végrehajtása közben

Tehát a (7+i*9) és (3+i*8) komplex számok összege (10+i*17). Igen ám, de mi nem a K+ szót szeretnénk használni az összeadásra! Sokkal jobb lenne csak simán a +, hiszen ezt szoktuk meg. Mivel csak komplexekkel akarunk dolgozni, nem zavar, hogy a + most azokra vonatkozik. Ha azután ismét egészekkel szeretnénk számolni, akkor majd megszabadulunk a + új jelentésétől. Tegyük tehát a következőt:

```
: + 4 ROLL ROT + ROT ROT + ;
```

Ha a definiciót valóban beirjuk egy FORTH rendszernek, figyelmeztető jelzést kapunk (aminek a szövege eltérhet az itt szereplőtől):

```
: + 4 ROLL ROT + ROT ROT + ; + redefined ok
```

A FORTH tehát először is jelzi az újradefiníció tényét, majd az "ok" üzenettel nyugtázza azt. Ezzel a + szó második, általunk megfogalmazott jelentése bekerül a szótárba és ettől fogva ez érvényes. Eközben természetesen régi jelentése sem veszett el, csak éppen nem hozzáférhető. Most már használhatjuk a K+ helyett:

```
7 9 3 8 + K. 10 17 ok
```

Figyeljük meg, hogy a + új definiciójában is van összeadás. Az ott szereplő + műveleteket a FORTH még az eredeti meghatározás szerint veszi figyelembe, azaz ekkor még egészekre vonatkoznak.

Az összeadáshoz hasonlóan járunk el a kivonásnál is:

```
: -          ( x1 y1 x2 y2 --- x1-x2 y1-y2 )
    4 ROLL   ( y1 x2 y2 x1 )
    ROT      ( y1 y2 x1 x2 )
    -        ( y1 y2 x1-x2 )
    ROT      ( y2 x1-x2 y1 )
    ROT      ( x1-x2 y1 y2 )
    -        ( x1-x2 y1-y2 ) ;
```

Természetesen a K. helyett is használhatjuk a .-ot, ugyanigy átdefiniálva. Ezzel a módszerrel a FORTH-t teljes egészében átalakithatjuk úgy, hogy komplex számokkal dolgozzon. Elkészithetjük a veremkezelő műveleteket, relációkat is. Egy komplex SWAP például igy nézne ki:

```
: SWAP          ( x1 y1 x2 y2 --- x2 y2 x1 y1 )
      4 ROLL ( y1 x2 y2 x1 )
      4 ROLL ( x2 y2 x1 y1 ) ;
```

Ha a szavak új definiciójától meg szeretnénk szabadulni és az előző meghatározását kívánjuk használni, egyszerűen elfelejtetjük a szót. Mivel a szótárból csak a legutolsó definició törlődik, az előzőek megmaradnak. Ha tehát komplex aritmetikánk elsőnek definiált szava a + volt, akkor a " FORGET + " hatására a teljes komplex aritmetika törlődik a szótárból.

Természetesen nemcsak az alapszótár szavait, hanem az általunk definiált szavakat is újradefiniálhatjuk. Egy szónévhez tetszőleges számú meghatározás tartozhat. A 3.5. ábra összefoglalja az újradefiniálásról eddig megismerteket.

```
: névl ... ;     1. definició
      .
      .          1. definició hatásköre
      .
: névl ... ;     2. definició
      .
      .          2. definició hatásköre
      .
FORGET névl      ez a 2. definiciót és a nála később
                 definiált szavakat törli. Innen az 1
                 definició érvényes.
```

3.5. ábra

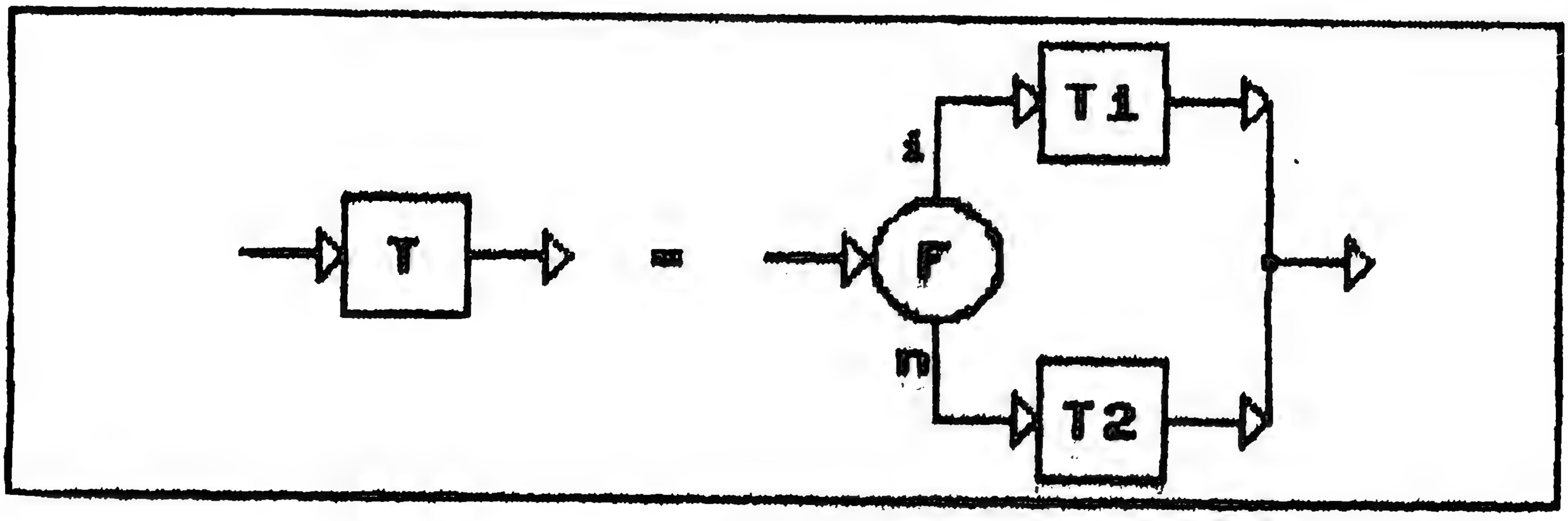
A többszörös definiciók hatáskörei

3.3. A programvezérlés eszközei

Eddig algoritmusainkat (szavainkat) csak egyféleképpen tudtuk felépiteni: már definiált részalgoritmusokat soroltunk fel, ezek sorrendben egymás utáni végrehajtását irtuk elő. Ezt a felépitési módot (összetételi eszközt) felsorolásnak nevezzük. Igy a T tevékenység a T(i) részfeladatok láncrafűzéséből áll.

Ez természetesen nem elégiti ki igényeinket. A FORTH tehát további összetételi eszközöket bocsát rendelkezésünkre. Ezek a választás és az ismétlés.

A választásnál egy feltételtől (F) függően a T1 vagy T2 résztevékenységet hajtjuk végre. (3.6. ábra)

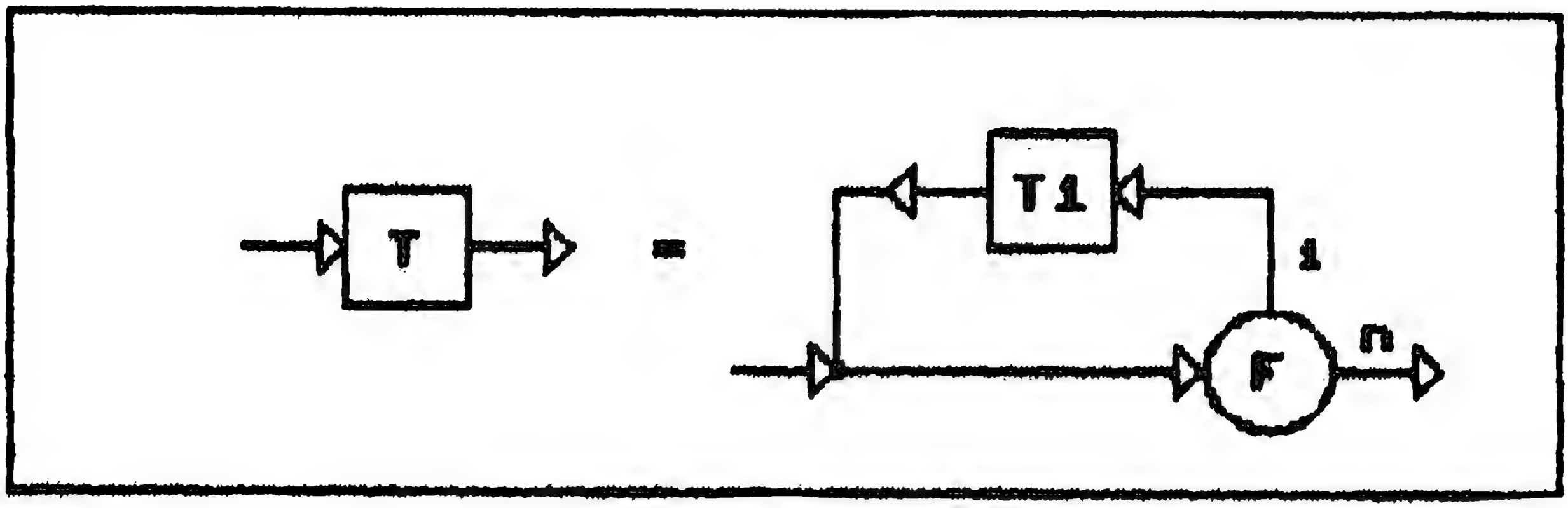


3.6. ábra
A választás folyamatábrája

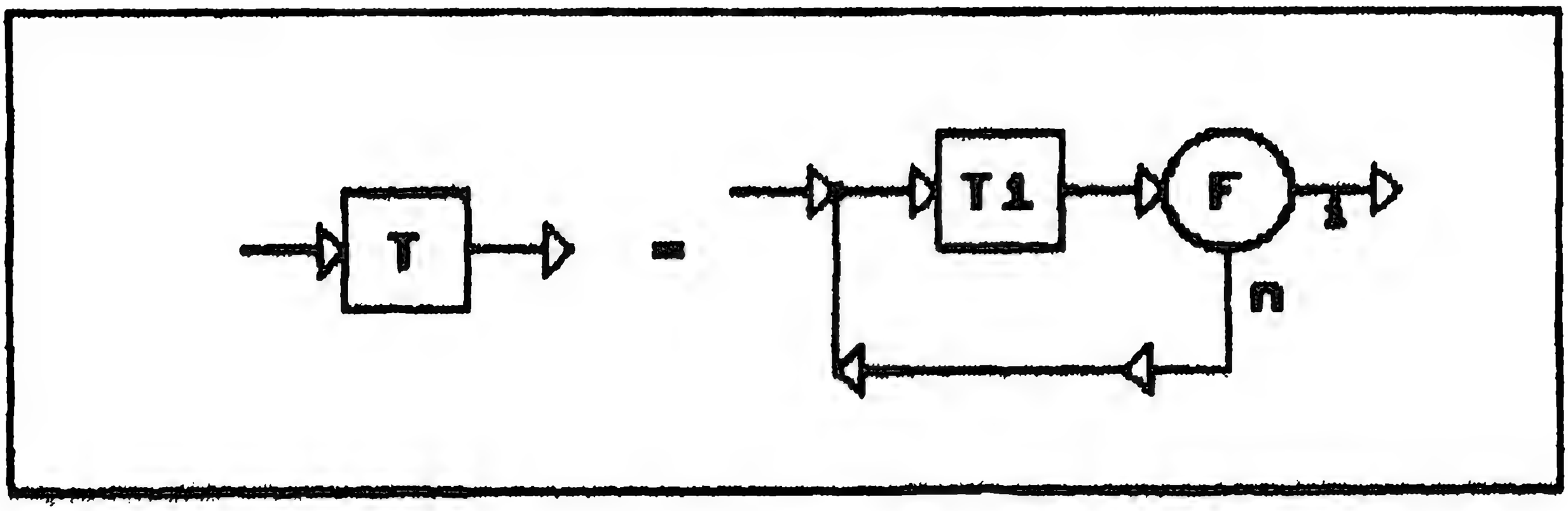
Ez a jól ismert IF..THEN..ELSE.. struktúra.

Az ismétlés lehetővé teszi a T1 tevékenység ciklikus végrehajtását. Azt, hogy az ismétlést folytassuk-e vagy sem, egy feltételtől tesszük függővé (3.7. ábra).

a)



b)



3.7. ábra
Az ismétlés folyamatábrája
WHILE (a) és UNTIL (b) esetén

Az a) eset a WHILE, a b) eset az UNTIL szerkezet. A végtelen ciklus az ismétlés speciális esete. Ekkor olyan feltételt irunk elő, hogy a ciklus sohase fejeződjön be. A szinte minden programozási nyelvben (igy a FORTH-ban is) megtalálható FOR ciklus szintén ismétlés, de itt a ciklus-

számláló növeléséről vagy csökkentéséről és a kilépési feltétel ellenőrzéséről az interpreter gondoskodik.

A FORTH tehát az összetételi eszközök közül a felsorolást, az ismétlést és a választást bocsátja rendelkezésünkre.

Vegyük észre, hogy a FORTH-ból hiányzik a GOTO vezérlésátadó utasitás! Igy még lehetőségünk sincs arra, hogy programunkban szabadon ugrálhassunk egyik helyről a másikra, megfosztva azt minden struktúráltságától. Ennek az utasitásnak a hiányát az értékelheti csak igazán, aki próbált már megérteni vagy javitani egy GOTO parancsoktól hemzsegő BASIC programot.

Lássuk ezek után a vezérlő utasitások konkrét alakját. Itt szeretnénk felhivni az Olvasó figyelmét arra, hogy a most következő utasitások csak szódefinició belsejében fordulhatnak elő. Ha e szavakkal interpreter módban próbálkozunk, a FORTH hibajelzést ad.

3.3.1. Vezérlőutasitások

3.3.1.1. IF..ELSE..ENDIF, IF..ELSE..THEN

Ennek szintaktikája a következő:

szósor1 IF szósor2 ELSE szósor3 ENDIF

Az ELSE ág el is maradhat:

szósor1 IF szósor2 ENDIF

Az ENDIF helyett használhatjuk a THEN szót is, a kettő teljesen egyenértékű. Az IF szót megelőző (szósor1-el jelölt) utasitássorozat végzi a feltételképzést. Ennek nem kell feltétlenül valódi feltételes utasitásokat (tehát

például 0<, =, 0= stb.) tartalmaznia. Lényeg az, hogy az IF végrehajtása előtt a vermen egy logikai értékként kezelhető szám maradjon. Az IF ugyanis a következőképpen működik: ha a verem tetején nullától különböző számot talál, akkor végrehajtódik az IF és az ELSE közötti utasitássorozat, majd a vezérlés az ENDIF utáni utasitásokra kerül. Ha a vermen nulla volt, akkor az ELSE és ENDIF közötti ág hajtódik végre. Az IF minden esetben leveszi a veremről az elágazást vezérlő számot.

Az ELSE ág nélküli utasitás működése értelemszerű: a vermen levő szám nemnulla értéke esetén az IF és ENDIF közötti szavak végrehajtódnak.

Nézzünk néhány egyszerű példát.

```
: ELOJELET.IR ( n --- )
                0< IF ." NEGATIV "
                   ELSE ." POZITIV " ENDIF ;
: ELOJEL ( n --- )
          DUP . ELOJELET.IR ;
```

ELOJEL szavunk a verem tetején levő számról eldönti, hogy az negativ vagy pozitiv-e, és ezt a számmal együtt ki is irja. A szó kipróbálása ezt eredményezi:

```
-2 ELOJEL -2 NEGATIV ok
73 ELOJEL 73 POZITIV ok
0 ELOJEL 0 POZITIV ok
```

Következő példánk nagybetű-kisbetű konverzió. Az ASCII karakterkészletben a nagybetűk (A..Z) kódja 65-től 90-ig, a kisbetűké (a..z) 97-től 122-ig terjed. Ha tehát a nagybetű kódjához 32-t hozzáadunk, a megfelelő kisbetűkódot kapjuk. Feladatunk, hogy a verem tetején levő számról eldöntsük, hogy nagybetűkód-e. Ha igen, kisbetűvé konvertáljuk, ha nem, változatlanul hagyjuk. A kapott kódot karakter formájában ki is iratjuk. A megoldás:

```
: KOZOTTE.VAN? ( n alsóhatár felsőhatár - jelző )
               SWAP ( n fh ah ) >R ( n fh )
               OVER ( n fh n ) < ( n fh<n )
               SWAP ( fh<n n )
               R>   ( fh<n n ah )
               <    ( fh<n n<ah )
               OR   ( fh<n VAGY n<ah )
               0=   ( jelző ) ;
: +32.HA.KELL ( n --- n+a, a=0 vagy 32 )
               DUP 65 90 KOZOTTE.VAN?
               IF 32 +
               ENDIF ;
: KONVERT ( n --- )
          DUP ( n --- n n )
          EMIT ( n n --- n )
          +32.HA.KELL ( n --- n+a )
          EMIT ( n+a --- ) ;
```

A KOZOTTE.VAN? akkor ad igaz logikai értéket, ha a vermen levő szám nagyobb-egyenlő az alsó határnál és kisebb-egyenlő a felsőnél, azaz benne van a megadott zárt intervallumban. Figyeljük meg a visszatérési verem használatát átmeneti adattárolásra! A +32.HA.KELL és a KONVERT működése nem szorul magyarázatra.

Próbáljuk ki a szót:

```
42 KONVERT ** ok
72 KONVERT Hh ok
73 KONVERT Ii ok
```

A * nem nagybetű, igy azt változatlanul kapjuk vissza. A H és I betűket viszont a szó a megfelelő kisbetűvé alakitotta át.

Utolsó példánk kissé bonyolultabb. A feladat a

következő: beolvasunk a billentyűzetről egy karaktert. Erről eldöntjük, hogy értelmezhető-e hexadecimális számjegyként, azaz a 0..9 vagy.A..F tartományba esik-e. Ha igen, akkor e számjegy decimális értékét (0..15), ha nem, egy kérdőjelet irunk ki. A decimális szám vagy kérdőjel előtt magát a beolvasott karaktert is kiiratjuk. Egy lehetséges megoldás a következő:

```
-1 CONSTANT IGAZ
0  CONSTANT HAMIS
: A..F->10..15 ( n1 --- n2 )
                        DUP 65 70 KOZOTTE.VAN?
                        IF 55 - IGAZ
                        ELSE HAMIS
                        ENDIF ;
: 0..F->0..15 ( n1 --- n2 )
                        DUP 48 57 KOZOTTE.VAN?
                        IF 48 - IGAZ
                        ELSE A..F->10..15
                             ENDIF ;
      : HEX->DEC ( --- )
                   KEY DUP EMIT SPACE
                   0..F->0..15 IF .
                               ELSE DROP ." ? "
                               ENDIF ;
```

A HEX->DEC szó kipróbálása a következő futtatási képet adja:

```
HEX->DEC A 10 ok
HEX->DEC 8 8 ok
HEX->DEC G ? ok
```

A konverzióhoz megirt három szó működésének megértését az Olvasóra bizzuk. Segitségképpen megjegyezzük, hogy a szavak megirásánál egy sűrűn használt programozói fogást

alkalmaztunk. Nevezetesen azt, hogy az A..F->10..15 és 0..F->0..15 szavak megfelelő logikai értékeket (IGAZ vagy HAMIS) helyeznek el a vermen attól függően, hogy feladatukat sikerült-e elvégezniük vagy sem. Az A..F->10..15 tehát IGAZ értéket (-1-et) tesz a veremre, ha a karakter az A..F tartományba esett, és igy át tudta alakitani a 10..15 decimális számmá. ha ez nem igy volt, akkor HAMIS (0) kerül a veremre. A 0..F->0..15 is hasonló logikával működik. Az igy képzett feltételeket azután a HEX->DEC IF utasitása használja fel a szám vagy a kérdőjel kiirásához. Ez egyben példa arra is, hogy az IF-et, mint azt már emlitettük, nem mindig előzi meg valódi relációs művelet. Az elágazásvezérlő feltételt ebben az esetben az A..F->10..15, ill. a 0..F->0..15 szolgáltatta.

Emellett érdemes megjegyezni, hogy az IF..ELSE..ENDIF szerkezetek tetszőleges mélységben egymásba skatulyázhatók, azaz az IF..ELSE és ELSE..ENDIF ágakban további feltételes elágazások (és más, a továbbiakban ismertetendő szerkezetek) lehetnek. Igy például a 0..F->0..15 szóban hivott A..F->10..15 helyére annak törzsét minden további nélkül beirhattuk volna. Az ilyen egymásba ágyazásokat mégis minden esetben, ha arra lehetőség van, kerüljük el! A többszintű szerkezeteket bontsuk fel különálló szavakra! Ezzel programjaink szerkezete világos és áttekinthető marad.

3.3.2. Ciklusszervező utasitások

A FORTH nyelv a ciklusszervező utasitások széles skáláját kinálja, igy lehetőség van arra, hogy az adott feladathoz leginkább illeszkedő tipust használjuk. A következő pontokban a rendelkezésünkre álló szerkezeteket fogjuk áttekinteni.

Itt szeretnénk megjegyezni, hogy a ciklusutasitások ugyanúgy, mint ahogy azt az IF-nél láttuk, tetszőlegesen egymásba ágyazhatók. Az ilyen többszintű ciklusok kiértékelése nem a FORTH-nak, hanem nekünk fog gondot okozni, amikor elveszünk a skatulyák szövevényében. A belső ciklusokat inkább emeljük ki különálló szavakba. (Kétszintű egymásba ágyazást még áttekinthetőnek tartunk).

3.3.2.1. DO..LOOP, +LOOP, /LOOP

Ez a más programozási nyelvekből (BASIC, PASCAL) jól ismert FOR..TO..STEP tipusú ciklus. A DO ciklus egy utasitássorozat adott számú végrehajtására alkalmas, miközben egy kitüntetett változó, az úgynevezett ciklusváltozó értéke folyamatosan nő vagy csökken. Szintaktikája a következő:

nvég nkezdő DO szósor LOOP

Ebben az esetben a ciklusváltozó a ciklus magjának minden egyes végrehajtása után eggyel növekszik. A ciklusváltozó létrehozásáról a FORTH gondoskodik, tehát azt kijelölnünk nem szükséges és nem is lehetséges. Az "nkezdő" a ciklusváltozó kezdőértéke, az "nvég" az az érték, amit a ciklusváltozó már nem vesz fel. Tehát a

```
10 0 DO ." forth" LOOP
```

utasitás tizszer kiirja a "forth" szót, miközben a ciklusváltozó rendre felveszi a 0,1,...,9 értékeket, de a 10-et már nem. A FORTH a ciklusváltozót és a végértéket a visszatérési vermen tárolja. A kilépési feltételt a LOOP szó ellenőrzi. Előbb egyet hozzáad a ciklusváltozóhoz, majd megvizsgálja, hogy az nagyobb-egyenlő-e az általunk megadott végértéknél. Ha igen, a vezérlés a LOOP utáni szóra adódik, ellenkező

esetben a DO...LOOP közötti ciklusmag újból végrehajtódik. Ebből következik, hogy a ciklus magja legalább egyszer mindig lefut, tehát nkezdő>=nvég kezdőparaméterek esetén is.

A DO előtti nvég és nkezdő nem feltétlenül két szám odairását jelenti. Hasonlóan az IF-hez, a DO előtt is tetszőleges szavak állhatnak, csak itt a DO elérésekor természetesen nem egy logikai értéknek, hanem két számnak kell lennie a vermen, amelyek nvég és nkezdőként fognak értelmeződni. Ugyanez igaz a most következő "nköz"-re is.

Ha azt szeretnénk, hogy a ciklus lépésköze ne egy, hanem egy általunk megadott szám legyen, a következő szerkezetet kell használnunk:

nvég nkezdő DO szósor nköz +LOOP

A LOOP helyett tehát +LOOP-ot irunk, és alkalmazzuk az "nköz" paramétert. Itt is érvényesek a DO..LOOP ciklusnál elmondott tulajdonságok, csak most a ciklusváltozóhoz egy helyett mindig az nköz fog hozzáadódni. A kilépési feltétel itt is ciklusváltozó>=nvég. Az elmondottakból következik, hogy az alábbi két ciklus teljesen egyenértékű:

```
22 7 DO ... LOOP
22 7 DO ... 1 +LOOP
```

A fenti két ciklus indexei a -32768...+32767 számtartományban mozoghatnak. A FORTH lehetőséget ad előjel nélküli egész számokkal szervezett ciklus használatára is, vagyis az indexhatárok 0...65535 tartományra való módositására. Erre szolgál a /LOOP utasitás. Az utasitás formátuma a DO...+LOOP-hoz hasonló:

uvég ukezdő DO szósor nköz /LOOP

A paraméterek jelentése a következő: ukezdő a ciklusváltozó kezdőértéke, uvég a végérték (mindkettő előjel nélküli egész), nköz a lépésköz (pozitiv szám az 1...32767 tartományban). Jegyezzük meg, hogy a /LOOP utasitást a FORTH gyorsabban hajtja végre, mint a +LOOP-ot.

Mindhárom fajta DO ciklus a ciklusmag első végrehajtása előtt a neki szükséges két paramétert leveszi a veremről és azokat a visszatérési veremre másolja. Ezért ezek a paraméterek a ciklus magjának és a ciklus környezetének adatforgalmát nem zavarják.

3.3.2.2. I,I',J,LEAVE

Már emlitettük, hogy a ciklusváltozó létrehozásáról a FORTH automatikusan gondoskodik. Az I, I', J szavak működését a 2.4.2. pontban tárgyaltuk. Hogy a változó aktuális értékéhez a ciklus magjában hozzáférhessünk, az I szót kell használnunk. Az I végrehajtásakor az éppen futó ciklus ciklusváltozójának értéke a veremre kerül. Az első öt páros számot például igy irathatjuk ki:

```
: PAROS 6 1 DO I 2* . LOOP ; ok
PAROS 2 4 6 8 10 ok
```

Ugyanezt persze igy is elérhetjük:

```
: PAROS 12 2 DO I . 2 +LOOP ; ok
PAROS 2 4 6 8 10 ok
```

Következő példánk legyen egy faktoriálisszámitó programrészlet (n!=1*2*...*n). Vegyük figyelembe, hogy definiciószerűen 0!=1. Ha paraméterként negativ számot adunk

meg, szavunk tegyen nullát a veremre. Mivel nincs olyan szám, amelynek faktoriálisa 0 lenne, ebből tudni fogjuk majd, hogy rossz bemenő paramétert adtunk meg. A feladat egy lehetséges megoldása:

```
: FAKTOR1 ( n --- n! )
          1     ( n 1 )
          SWAP  ( 1 n )
          1+    ( 1 n+1 )
          1     ( 1 n+1 1 )
          DO
           I    ( 1*2*...*[I-1] I )
           *    ( 1*2*...*[I-1]*I )
          LOOP ;
: FAKTOR2 ( n --- n!, n>=0 )
          ?DUP IF FAKTOR1
               ELSE 1
               ENDIF ;
: N! ( n --- n! )
     DUP 0< IF DROP 0
            ELSE FAKTOR2
            ENDIF ;
```

Ha N!-t kipróbáljuk, a következőt kapjuk:

```
-9 N! . 0 ok
3 N! . 6 ok
0 N! . 1 ok
```

A szavak működésének megértését az Olvasóra bizzuk, itt csak néhány támpontot adunk. FAKTOR1 szolgál a nullánál nagyobb számok faktoriálisának kiszámitására. A veremre tett 1 az 1*2*...*n szorzat első tényezője. Az 1+ azért kell, hogy a ciklus még felvegye n értékét is. (Hisz különben a ciklus egytől (n-1)-ig futna csak). FAKTOR2 szűri ki az n=0 esetet. Végül N! negativ számokra hibát jelez a nulla veremre

tevésével, nemnegativokra pedig meghivja FAKTOR2-t.

Egyes esetekben szükségünk lehet a ciklus magjában a ciklus végértékére is. Ezt az I' szó használatával érhetjük el. Az I'-t már megismertük a visszatérési vermet kezelő utasitásoknál (l. a 2.4.2. pontot). Mivel a szó a visszatérési verem második szintjén levő számot teszi az adatveremre, kiválóan alkalmas a ciklusvégérték elővételére, hisz az is ott tárolódik. Következő példánk egy "visszaszámláló" szó:

```
: VISSZASZAM 0 DO I' I - . LOOP ; ok
10 VISSZASZAM 10 9 8 7 6 5 4 3 2 1 ok
```

Lehetőség van arra is, hogy egymásba ágyazott ciklusok esetén a belső ciklusban hozzáférjünk a külső ciklus ciklusváltozójának aktuális értékéhez. Erre szolgál a J szó. Lássunk egy példát, amely az egyszeregy táblázatos kiirását végzi:

```
: EGYSZEREGY 11 1 DO CR
                    11 1 DO J I * 4 U.R
                          LOOP
                 LOOP ; ok
EGYSZEREGY
   1   2   3   4   5   6   7   8   9  10
   2   4   6   8  10  12  14  16  18  20
   3   6   9  12  15  18  21  24  27  30
   4   8  12  16  20  24  28  32  36  40
   5  10  15  20  25  30  35  40  45  50
   6  12  18  24  30  36  42  48  54  60
   7  14  21  28  35  42  49  56  63  70
   8  16  24  32  40  48  56  64  72  80
   9  18  27  36  45  54  63  72  81  90
  10  20  30  40  50  60  70  80  90 100 ok
```

3.3. A programvezérlés eszközei

Gyakran szükségünk van arra, hogy egy DO ciklust idő előtt elhagyjunk. Ezt legtöbbször akkor szoktuk alkalmazni, ha futás közben valamilyen hibát észlelünk és ezért a ciklusból ki szeretnénk lépni. A FORTH-ban ezt a ciklusmagban megfelelő helyen elhelyezett **LEAVE** (= hagyd el) utasitással érjük el. A LEAVE a következőképpen működik: a visszatérési vermen tárolt végértéket a ciklusváltozó aktuális értékére módositja. Emiatt, ha a ciklusmagban végrehajtódott egy LEAVE, akkor a LOOP, _OOP vagy a /LOOP végrehajtásakor a kilépési feltétel azonnal teljesül (hiszen nvég=I) és ez a ciklus befejezését jelenti. Tehát a LEAVE és a LOOP (+LOOP, /LOOP) közötti ciklusmagrészlet (ha van) még végrehajtásra kerül.

Lássunk egy egyszerű példát! Tegyük fel, hogy a vermen n darab karakter ASCII kódja van, valamint a verem tetején maga az n is megtalálható. Ezt az n darab karaktert szeretnénk kiiratni. Ez igen könnyű, máris irhatjuk a megoldást:

```
: KIIR 0 DO EMIT LOOP ;
```

Módositsuk most a feladatot úgy, hogy ha a karakterek között az EOT (End Of Text = szöveg vége) is előfordul, amelynek ASCII kódja 3, akkor a kiiratást be kell fejezni. Szavunk ekkor igy módosul:

```
: KIIR 0 DO DUP 3 = IF LEAVE ELSE EMIT ENDIF LOOP ;
```

A példa önmagáért beszél.

3.3.2.3. BEGIN..WHILE..REPEAT

A WHILE tipusu ciklusutasitás alakja:

BEGIN szósor1 WHILE szósor2 REPEAT

A ciklus a következőképpen működik: először a BEGIN és WHILE között levő utasitások hajtódnak végre. Ennek egy számot kell a verem tetején eredményezni, amelyet a WHILE logikai értékként fog kezelni (azaz a nulla a hamis, a nem nulla az igaz). Ha az érték igaz, akkor végrehajtódnak a WHILE és REPEAT közötti utasitások, majd a vezérlés újból a BEGIN utáni utasitássorozatra adódik. Ez a feltételkifejezés újbóli kiértékelését jelenti. Az eljárás mindaddig folytatódik, amig a WHILE hamis értéket nem talál a vermen, ami a ciklusból való kilépést eredményezi. A WHILE szó a ciklust vezérlő feltétel értéket mindig leveszi a veremről. Ha az első kiértékelésnél a feltétel hamis, a ciklusmag egyszer sem hajtódik végre.

Alábbi példánkban a BEKI szó karaktereket olvas be a billentyűzetről, ezeket ki is irja mindaddig, amig a beolvasott karakter nem carriage-return (= kocsi-vissza, ASCII kódja 13).

```
: BEKI ( --- )
       BEGIN KEY DUP 13 - ( kód kód<>13 )
       WHILE  ( kód ) EMIT
       REPEAT ( kód ) DROP ;
```

A következő szó az ASCII karakterkészlet egy általunk megadható intervallumát irja ki:

```
: ASCII ( alsó felső --- )
        1+ SWAP BEGIN OVER OVER >
```

```
                    WHILE DUP EMIT 1+
                    REPEAT DROP DROP ;
 65 70 ASCII ABCDEF ok
```

3.3.2.4. BEGIN..UNTIL, BEGIN..END

Ennek a ciklusszervező utasitásnak az alakja:

BEGIN szósor UNTIL

vagy

BEGIN szósor END

A kettő teljesen egyenértékű. A BEGIN és UNTIL között tetszőleges utasitásokat helyezhetünk el. Az egyetlen megkötés (mint azt az IF és a WHILE esetében már láttuk), hogy a UNTIL végrehajtása előtt a vermen egy logikai érték legyen. A ciklúsmag ismétlése mindaddig folytatódik, amig ez az érték hamis. A WHILE ciklussal ellentétben itt a BEGIN és UNTIL közötti rész (a kilépési feltétel ciklusvégi vizsgálatából következően) legalább egyszer mindig lefut.

Gyakorlásképpen készitsünk néhány, a BEGIN..UNTIL szerkezetet használó szót! Az előző pontban definiált BEKI-t irjuk át úgy, hogy most az UNTIL tipusú ciklust használja:

```
: BEKI ( --- )
   BEGIN KEY ( --- n )
         DUP ( n --- n n )
         EMIT ( n n --- n )
         13 = ( n --- f )
   UNTIL ;
```

Vegyük észre, hogy ez a BEKI nem pontosan úgy működik, mint azt az elődje tette! Ez ugyanis a soremelés karaktert is kiirja a begépelt szöveg után, az előző viszont nem. A BEKI-t kicsit átalakitva most készitsük el a BEVESZ szót, amely a beolvasott szöveget nemcsak kiirja, de PAD-tól kezdve a memóriába is elteszi.

```
: BEVESZ            ( --- )
PAD 1-              ( ---  PAD-1 / = cim / )
BEGIN
    1+              ( cim --- cim+1 )
    KEY             ( cim --- cim n )
    DUP DUP         ( cim n --- cim n n n )
    EMIT            ( cim n n n --- cim n n )
    3 PICK          ( cim n n --- cim n n cim )
    C!              ( cim n n cim --- cim n )
    13 =            ( cim n --- cim f )
UNTIL
DROP                ( cim --- ) ;
```

Irjuk meg az SZOKOZ nevű szavunkat, amely az eltárolt szövegben levő szóközök számát fogja a SZKSZAM nevű változóba tenni! A megoldás igen egyszerű. Itt is UNTIL tipusú ciklust fogunk használni, a szöveg végét a kocsi-vissza karakter jelzi majd.

```
VARIABLE SZKSZAM
: SZOKOZ 0 SZKSZAM !
         PAD 1- BEGIN 1+ DUP C@ DUP 32 =
                   IF 1 SZKSZAM +!
                   ENDIF 13 =
         UNTIL DROP ;
```

Szavainkat igy használhatjuk:

BEVESZ<R> mint a mókus fenn a fán<R> ok

SZOKOZ SZKSZAM ? 5 ok

3.3.2.5. BEGIN..AGAIN

A BEGIN és AGAIN közé tetszőleges utasitássorozatot irhatunk:

BEGIN szósor AGAIN

Ez a végtelen ciklus, amelyből természetes úton kilépni nem lehet. Például a

: VEGTELEN BEGIN KEY EMIT AGAIN ;

definició után a VEGTELEN szó végrehajtásával az idők végezetéig (de legalábbis a számitógép kikapcsolásáig) irogathatunk karaktereket a képernyőre.

A végtelen ciklust gyakran használjuk egy nagyobb programrendszer legmagasabb szintű vezérlőszerkezeteként. Valójában a FORTH interpretere is egy végtelen ciklus. Előbb beolvassa a forrássort, azt feldolgozza és a szavakat végrenajtja. Ezután kiirja az "ok" üzenetet, majd kezdődik az egész elölről.

3.3.3. Rekurzió

Ha egy szó önmagát hivja meg, rekuzióról beszélünk. Bár a rekurzióra a FORTH-ban lehetőségünk van, általában ritkán alkalmazzuk. Ha mégis e módszer mellett döntünk, akkor szavainkat mindig a legnagyobb figyelemmel tervezzük meg. A rekurziós szavak a veremszintek legnagyobb fogyasztói. A

felhasznált szintek száma mind az adat-, mind a visszatérési vermen minimum a hivási mélységgel egyezik meg. Ez jól működő szó esetén nem okoz gondot (a vermek mérete általában 1...4 kbájt), de a programozók nagy előszeretettel készitenek végtelen rekurziókat. Ilyen esetben ne csodálkozzunk, ha a rossz szó jobb hiján a szótárterületet is veremnek használja, majd a rendszer "elszáll". Ezért mindig jól gondoljuk át a rekurziv szó működését, biztositsuk a megfelelő kilépési feltételeket. Könyvünk kereteit meghaladná annak tárgyalása, hogy egy rekurziv formula mikor helyettesithető valamilyen ciklusutasitással, de az esetek többségében ez igy van. A ciklussá való átfogalmazás a matematikai szabályok ismerete nélkül, józan logikával is megtehető.

A fentiekkel nem lebeszélni akartuk az Olvasót a rekurzióról, mindössze a módszerben rejlő csapdákra szerettük volna felhivni a figyelmet.

Még a szabványosnak nevezett FORTH implementációk sem egységesek abban, hogy miként biztositják a rekurziót. Két módszer ismeretes, közülük valamelyik, esetleg mindkettő is biztos megtalálható egy adott megvalósitásban. A kettő között elvi különbség nincs, inkább jelölésbeli eltérésről beszélhetünk.

3.3.3.1. SMUDGE

A szódefiniciók szótárbeli feje tartalmaz egy úgynevezett smudge-bitet (smudge = piszok, folt). Ha ez a bit egy, akkor a FORTH a szótárbeli kereséskor az adott szót figyelmen kivül hagyja, azaz nem hajlandó felismerni. Ez a bit szódefinició közben mindaddig egyes marad, amig a szó forditása sikeresen be nem fejeződik. Ezért nem irhatjuk le, hogy

: NÉV ... NÉV ... ;

mert a NÉV definiciója közben (a smudge-bit egy értéke miatt) maga a NÉV még nem ismert. Hogy ezt a bitet nullára állitsuk, a **SMUDGE** szót kell használnunk. A SMUDGE a bitet mindig ellenkezőjére váltja, azaz ha egy volt, akkor nulla lesz és forditva. Nézzünk használatára egy példát! A 3.3.2.2. pontban már készitettünk egy N! nevű faktoriálisszámitó szót. Irjuk meg az N!-t most a rekurzió felhasználásával. A faktoriálisszámitás rekurziv képlete a következő:

n=0 ==> n!=1
n>0 ==> n!=n*(n-1)!

A megoldás egyszerű, mindössze a fenti összefüggéseket kell FORTH-ban megfogalmazni (a negativ számokat, mint lehetséges bemenő paramétereket hagyjuk figyelmen kivül):

```
: N! ( n --- n! ; n>=0 )
[ SMUDGE ] ?DUP
           IF DUP 1- ( n --- n n-1 )
              N!     ( n n-1 --- n (n-1)! )
              *      ( n (n-1)! --- n! )
           ELSE 1    ( 0! )
           ENDIF ; SMUDGE
```

Az N! szó elején kiadjuk a SMUDGE utasitást, mégpedig szögletes zárójelek közé irva. A [és] szavakról a 3.5. szakaszban szólunk részletesen, most elégedjünk meg annyival, hogy a közéjük irt szó forditás közben hajtódik végre, ahelyett, hogy cime a szótárba kerülne. Igy a SMUDGE egyről nullára állitja a smudge-bitet, a FORTH tehát az N! definiálása közben is fel fogja ismerni a szó nevét. Mivel a definició végén a ; is kiad egy SMUDGE-ot (normális esetben emiatt válik ismertté a szó), a bit most megint egy lesz, ami ismeretlen szóvá teszi az N!-t. Ezért a ; után még egy

SMUDGE szükséges a bit újbóli nullára állitása érdekében.

3.3.3.2. MYSELF

Egyes FORTH implementációk készitői azt vallják, hogy nem szerencsés, ha a smudge-bit átállitásával egy definiálás alatt álló (és esetleg szintaktikusan hibás) szó ismertté válik az interpreter számára. Ezért a rekurzióhoz bevezetik a **MYSELF** (= önmagam) szót, amelyet a rekurziv hivásban a szó neve helyett kell alkalmazni. A MYSELF nem a smudge-bitet változtatja meg, hanem kiszámitja és elhelyezi a szótárban annak a szónak a cimét, amit éppen helyettesit. További magyarázatok helyett közöljük az N! MYSELF-es alakját:

```
: N! ( n --- n! ; n>=0 )
                ?DUP
                IF DUP 1- ( n --- n n-1 )
                   MYSELF ( n n-1 --- n (n-1)! )
                   *      ( n (n-1)! --- n! )
                ELSE 1    ( 0! )
                ENDIF ;
```

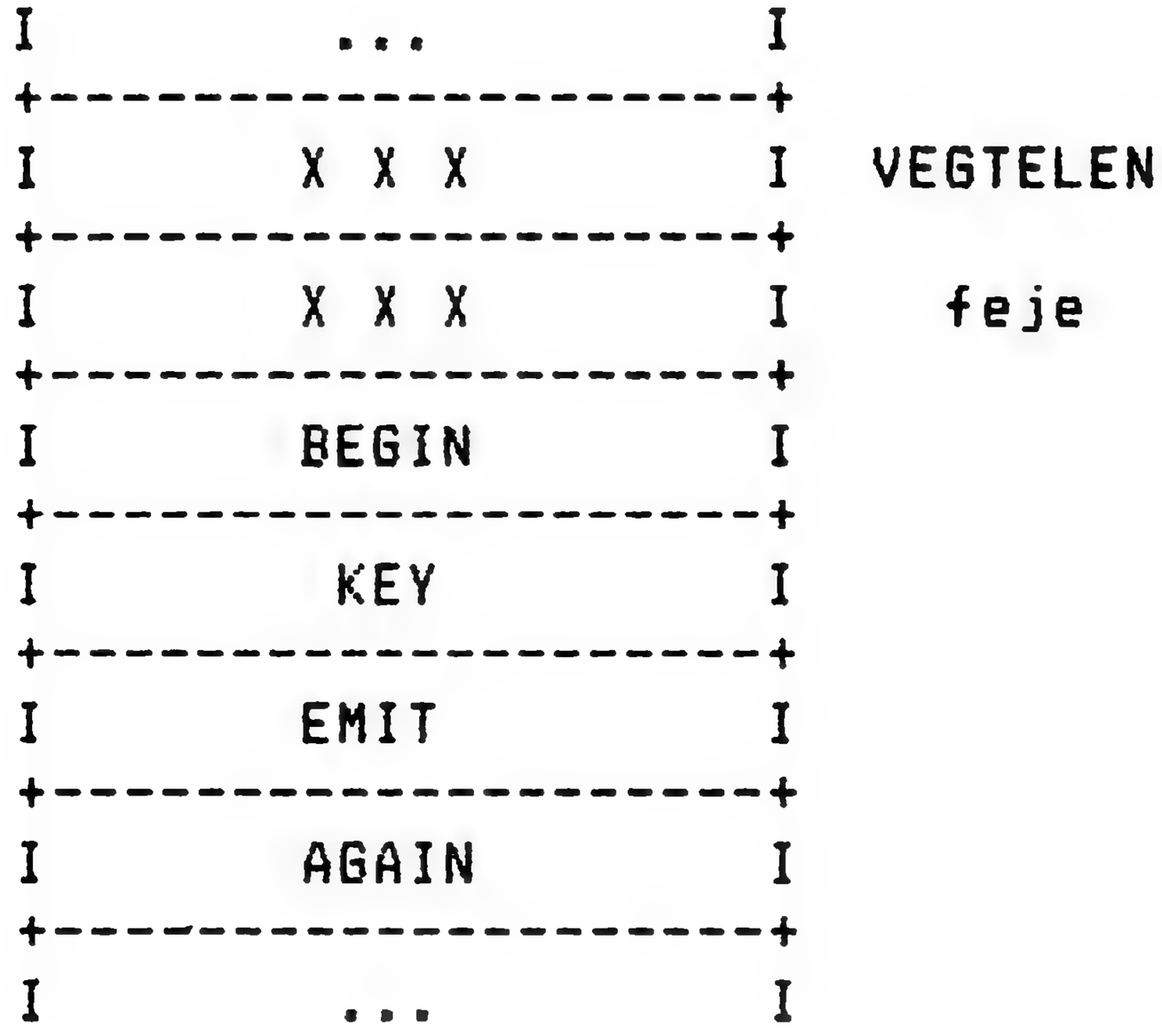
A különbség jól látható: elhagyjuk a SMUDGE-okat és a törzsben az N! hivás helyére MYSELF-et irunk.

A két módszer teljes mértékben egyenértékű a rekurziv szó működése szempontjából.

3.4. Közvetlen szavak

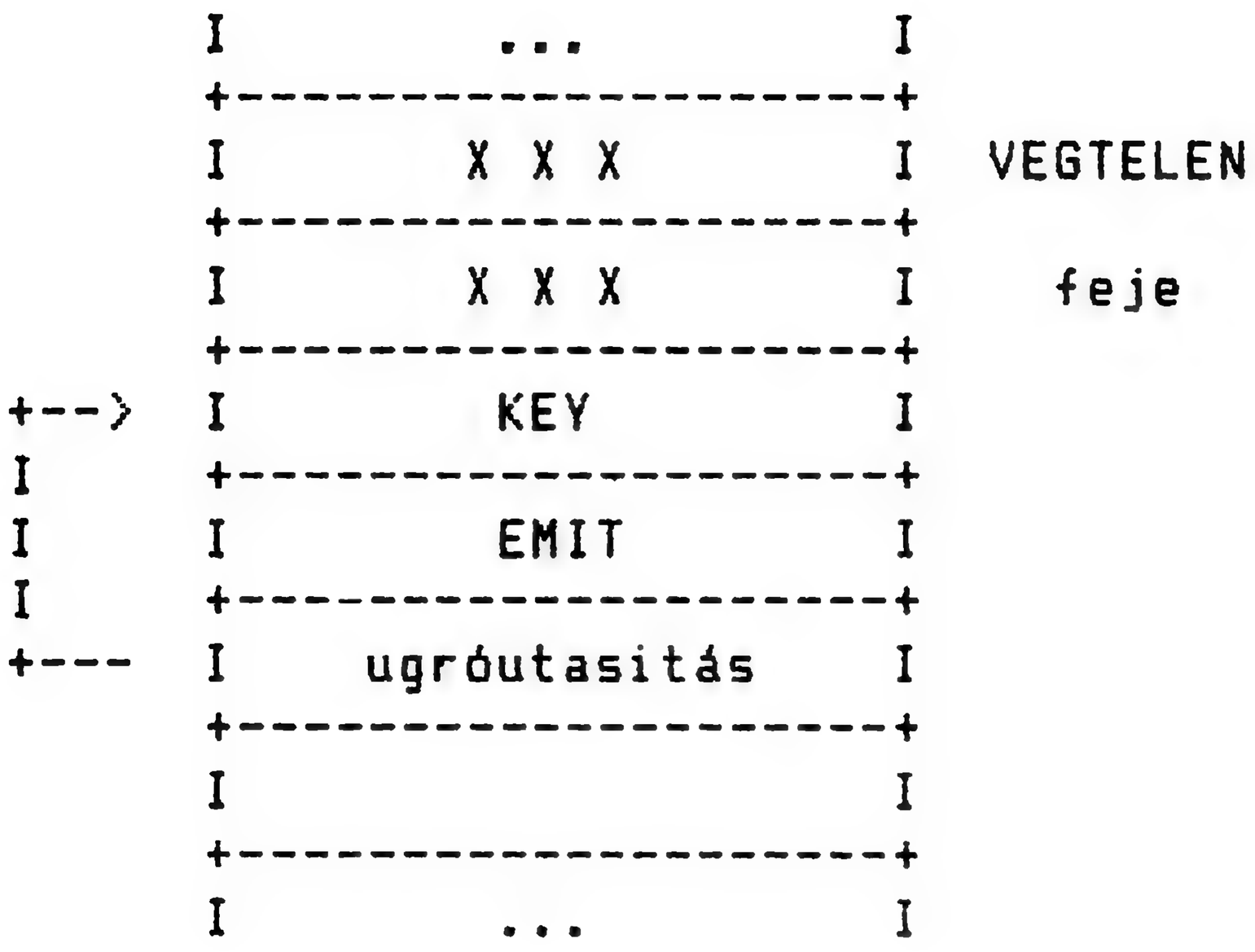
A 3.2. szakaszban beszéltünk arról, hogy miképpen

működik a FORTH compiler üzemmódban. Ha visszaemlékezünk rá, a lényeg az volt, hogy az új szó törzsében felsorolt szavak cimeit elhelyezi a szótárban. Ha ez igaz, akkor viszont nyugodtan mondhatjuk azt, hogy a 3.3.2.5. pontban bemutatott VEGTELEN szó a szótárban a 3.8. ábra szerint néz ki.



3.8. ábra
Eddigi ismereteink alapján ilyennek gondolnánk a VEGTELEN törzsét

Az ábrát nézve a legtöbb Olvasóban valószinüleg felmerül a kérdés, hogy vajon a végrehajtás közben az AGAIN hogyan talál vissza a BEGIN utáni utasitásra, a KEY-re? (A KEY a ciklusmag első utasitása). Talán addig keres a szótárban visszafelé, amig a BEGIN cimét meg nem találja? Látszik, hogy ez nem lenne valami jó megoldás. Esetleg az VEGTELEN egészen másképp néz ki a szótárban? Igy igaz (3.9. ábra).



3.9. ábra
Valójában ilyen a VEGTELEN törzse

A valóságban sem a BEGIN, sem az AGAIN cime nincs ott. Az AGAIN helyén egy ugróutasitás, a BEGIN helyén pedig semmi sincs. Ugyanigy nem találnánk meg a szótárban az IF-et, az ELSE-t, a DO-t, sem a többi hasonló utasitást (pontosabban a cimüket). Ez viszont csak úgy lehetséges, ha ezek a szavak valamiképp befolyásolják a FORTH működését compiler üzemmódban. Teszik ezt oly módon, hogy forditás állapotban is végrehajtódnak. A BEGIN például forditás közben egy cimet tesz a veremre, az AGAIN ezt a cimet felhasználva egy ugróutasitást szerkeszt a szótárba. Hasonlóképpen működik az UNTIL vagy a REPEAT is. Igy már érthetőbb a dolog, de még mindig kérdés, hogy miről ismeri fel a FORTH, hogy vajon beforditani vagy végrehajtani kell-e egy szót?

Tudjuk, hogy a szótár egy-egy szóról többféle

információt tartalmaz. Többek közt egy úgynevezett precedenciabitet, ami megmondja a FORTH-nak, hogy mit kell tennie. Ha a bit nulla, akkor a FORTH a szó cimét elhelyezi a szótárban. Ez a szokásos fordításidejű működés. Ha viszont ez a bit egyes, akkor a szó (függetlenül attól, hogy interpreter vagy compiler módban vagyunk-e) azonnal végrehajtódik.

Ilyen tulajdonságú szavakat magunk is készithetünk, az **IMMEDIATE** (= azonnal, rögtön) szó felhasználásával. Lássunk egy példát:

```
: UDV ." hogy vagy? " ; IMMEDIATE
```

Figyeljük meg, hogy az IMMEDIATE helye nem a szódefinició belsejében (a : és ; között) van, hanem a ; után. A FORTH-ban az ilyen, fordítás közben is végrehajtódó szavakat **közvetlen**nek nevezzük. Ha most kipróbáljuk a szót, semmi különlegeset nem veszünk észre. Ugyanúgy működik, mint az eddig megismertek.

```
UDV hogy vagy? ok
```

Most irjuk be az UDV-öt egy szódefinicióba:

```
: NEV ." a nevem " UDV ." forth " ; hogy vagy? ok
```

Rögtön látjuk a különbséget: az UDV végrehajtódott, ahelyett hogy cime a NEV szótárbeli meghatározásába került volna. Ezt bizonyitja a "hogy vagy?" üzenet. Ha a NEV szót kipróbáljuk, az UDV-nek nyomát sem fogjuk találni benne:

```
NEV a nevem forth ok
```

Foglaljuk tehát össze, mit tudunk az UDV-ről:

1. Interpreter módban úgy működött, mint bármely más szó.

2. Cómpiler módban (azaz szódefinició belsejében) azonnal végrehajtódott.

3. Az őt tartalmazó szóra annak futásakor semmilyen hatással nem volt.

Hasonló tulajdonságokkal rendelkezik a BEGIN is, ami egy cimet tesz a veremre. Nem mondhatjuk el ugyanezt már az AGAIN-ról, amire a 3. tulajdonság nem teljesül. Az AGAIN ugyanis úgy befolyásolja az őt tartalmazó szót, hogy annak szótárbeli meghatározás részébe egy ugróutasitást fordit. Emiatt a szó végrehajtásakor ezt elérve újra kezdődik a ciklusmag ismétlése. Hogyan tudta ezt az AGAIN megtenni? Erre ad választ a következő szakasz.

3.5. Forditásvezérlő utasitások

3.5.1. COMPILE

Azt szeretnénk tehát megtudni, hogy egy közvetlen szó hogyan képes futása közben az őt tartalmazó szó testében más szócimeket elhelyezni. Ezt a következő példa segitségével érthetjük meg.

```
: HUHU ." huhu " ;
```

Ez egy teljesen mindennapos szó, ha lefuttatjuk, kiirja azt, hogy huhu, ha szódefinició belsejében van, akkor cime a szótárba kerül. Lássuk most a következő szót:

3.5. Forditásvezérlő utasitások

```
: UJHUHU ." forditom a HUHU-t "
          COMPILE HUHU ; IMMEDIATE
```

Ebben egy új szóval találkozunk. Ez a **COMPILE** (= forditsd be). A COMPILE általában közvetlen szavakban fordul elő. Hatására az közvetlen szó végrehajtásakor a COMPILE-t követő szó (példánkban a HUHU) cime bekerül a szótárba. Ha most az UJHUHU-t felhasználjuk, a következő eredményt kapjuk:

```
: DATUM ." 1961.03.31 " UJHUHU ; forditom a HUHU-t ok
DATUM 1961.03.31 huhu ok
```

További magyarázatok helyett nézzük, mi lenne, ha az UJHUHU nem tartalmazná a COMPILE szót:

```
: UJHUHU CR ." forditom a HUHU-t "
          HUHU ; IMMEDIATE ok
: DATUM ." 1961.03.31 " UJHUHU ;
 forditom a HUHU-t huhu ok
DATUM 1961.03.31 ok
```

A COMPILE szó segitségével helyezi el a megfelelő feltételes vagy feltétlen ugróutasitásokat a szótárban az AGAIN, UNTIL, REPEAT és a LOOP is. Próbálja meg az Olvasó eddigi ismeretei alapján kikövetkeztetni, hogy például a DO szó mit fordithat a szótárba a COMFILE felhasználásával? (Válasz: egy olyan, egyébként (DO)-nak nevezett szót, amely az adatveremről a visszatérési veremre másolja a ciklusvezérlő paramétereket)

3.5.2. [COMPILE]

Most már majdnem mindent tudunk a közvetlen szavakról, kivéve azt, hogy mit tegyünk, ha mégsem akarjuk az azonnali végrehajtást. Például szeretnénk, ha a 3.4. szakaszban megirt NEV szó szótárbeli meghatározásába kerüljön be az UDV cime. A FORTH-ban erre is van lehetőség, mégpedig a [COMPILE] szó használatával. Lássuk az új NEV-et:

```
: NEV ." a nevem forth " [COMPILE] UDV ; ok
```

Vegyük észre, hogy bár az UDV közvetlen szó, fordítás közben nem hajtódott végre, hisz nem jelent meg a "hogy vagy?" üzenet. Az új NEV-et kipróbálva ezt látjuk:

```
NEV a nevem forth hogy vagy? ok
```

A [COMPILE] maga is közvetlen szó. Működésének lényege, hogy szódefinició közben végrehajtódva mintegy ellopja a FORTH elől a következő közvetlen szót, előállitja cimét és azt a szótárban elhelyezi. Igy nyilik lehetőségünk arra, hogy egy szó végrehajtás helyett mégiscsak fordításra kerüljön.

Ne keverjük össze a kétféle COMPILE-t!

1. A COMPILE általában közvetlen szavakban fordul elő, és közönséges szavak törzsében helyez el cimeket a közvetlen szó futásakor.

2. A [COMPILE] általában közönséges szavakban fordul elő és közvetlen szavak cimét tudja a törzsben elhelyezni fordítás közben.

3.5.3. A [és a]

A [és] szavak (nyitó és záró szögletes zárójel) csak szódefinició belsejében fordulhat elő. A [felfüggeszti a compiler módot és interpreter módba kapcsol. A] visszakapcsol compiler módba. Lássuk, mire jó mindez. Példaként használjuk megint a NEV szót.

```
: NEV ." a nevem forth" [ ." huhu " ] ; huhu ok
```

A [és] közötti rész tehát végrehajtódott, hiszen akkor interpreter módban voltunk. A NEV futtatása ezt eredményezi:

```
NEV a nevem forth ok
```

A szögletes zárójelek közötti utasitás a NEV működésére hatástalan. A bemutatott példából is látható, hogy a [és] esetén a közvetlen szavakhoz hasonló helyzettel találkozunk. Emlékezzünk vissza a rekurzióra (l. a 3.3.3. pontban)! Ott már használtuk a szögletes zárójeleket, igy érve el a SMUDGE szótörzsbeli végrehajtását. A következő pontban a [és a] alkalmazására további példákat is látunk majd.

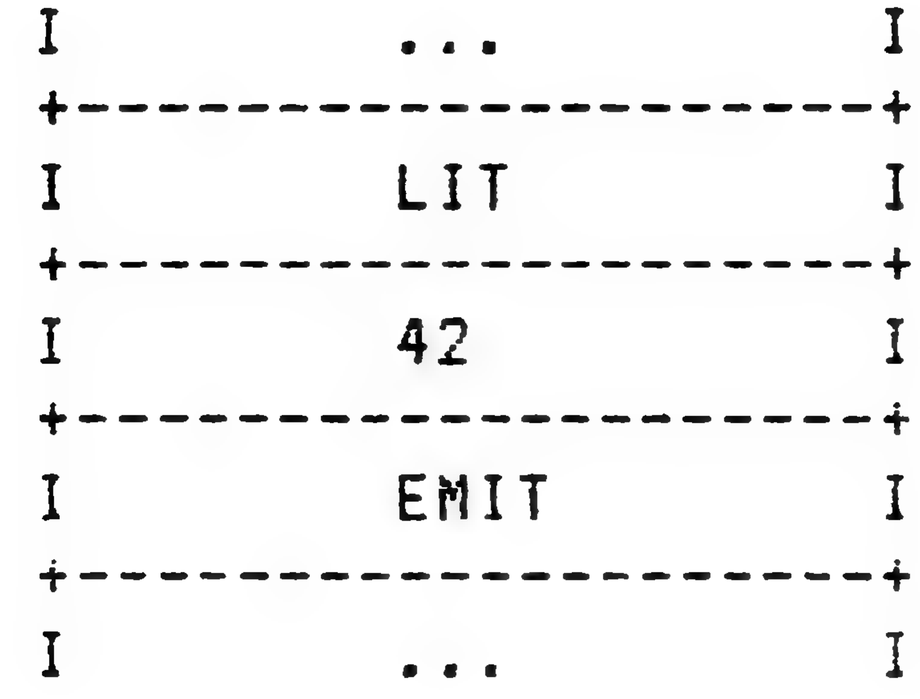
3.5.4. LIT, LITERAL, DLITERAL

Tekintsük a következő szót:

```
: CSILLAG 42 EMIT ;
```

A CSILLAG a 42 ASCII kódú * karaktert fogja a képernyőre kiirni. Sok Olvasóban talán már megfogalmazódott a kérdés, vajon hogyan tárolja a FORTH a számkonstansokat?

Minden bizonnyal nem úgy, hogy a szám önmagában áll a szótárban, hiszen akkor mi különbözteti meg a cimektől? Hogyan jön rá a FORTH arra, hogy a 42-t a veremre kell majd tenni a CSILLAG meghivásakor, és nem a 42-es cimen kezdődő szót kell végrehajtania? A megoldás a 3.10. ábrán látható.



3.10. ábra
A CSILLAG törzse. A számot a LIT előzi meg

Ha a FORTH forditás közben egy számot talál, akkor előbb elhelyezi a szótárban a **LIT** szó cimét, majd a következő memóriarekeszbe beirja a számot. (A LIT a literal szóból ered, jelentése: szó szerint. Azaz a LIT-et követő értéket nem cimként, hanem számként kell értelmezni.) Jegyezzük meg, hogy a LIT-et a FORTH automatikusan beirja a szótárba a számkonstansok előtt! Most nézzük meg, mi a teendő akkor, ha azt szeretnénk, hogy ne egy szótörzsben leirt számkonstans kerüljön a szótárba, hanem egy vermen levő szám!

Tegyük fel, hogy feladatunk egy 80 karakter széles képernyőre három sornyi csillagot kinyomtatni. Ezt igy oldhatjuk meg:

```
: CSILLAGOK 80 3 * 0 DO ." *" LOOP ;
```

A CSILLAGOK szó kiirja a 240 darab csillagot, de a FORTH-nak a szó minden meghivásakor ki kell értékelnie a "80 3 *" kifejezést. Ez növeli a futási időt, emellett felesleges is, hiszen 80*3=240. Irhatnánk tehát azt is, hogy

```
: CSILLAGOK 240 0 DO ." *" LOOP ;
```

Viszont itt a 240 nem sokat mond. A "80 3 *"-ban benne van az, hogy 80 karakter, 3 sor. A program olvashatóságát sokkal jobban szolgálja ez a forma. Melyiket válasszuk: a gyorsabbat vagy a kifejezőbbet? Egyiket sem! Van ugyanis egy harmadik megoldás, ami gyors és kifejező is.

```
: CSILLAGOK [ 80 3 * ] LITERAL 0 DO ." *" LOOP ;
```

Nézzük, hogyan működik ez a változat! A "80 3 *" szögletes zárójelek között van, azaz a CSILLAGOK forditása közben végrehajtódik. Emiatt a veremre kerül a 240. A következő szó a **LITERAL**. A LITERAL egy közvetlen szó, amely a szótárba forditja a LIT cimét, majd a LIT után elhelyezi a verem tetején található számot. Esetünkben tehát a LIT után most a 240 kerül a szótárba. Mit jelent ez? Futási időben nem vesz el időt a szorzat kiszámitása, mert az már forditáskor megtörtént. A programszöveget viszont a "80 3 *" érthetőbbé és logikusabbá teszi. A [,] párt és a LITERAL-t egyébiránt főleg a tárgyalt struktúra megvalósitására szoktuk használni. Ne feledjük tehát, hogy a

```
... [ számitás ] LITERAL ...
```

utasitássorozatot érdemes minél többször alkalmazni, ezzel is növelve programjaink áttekinthetőségét.

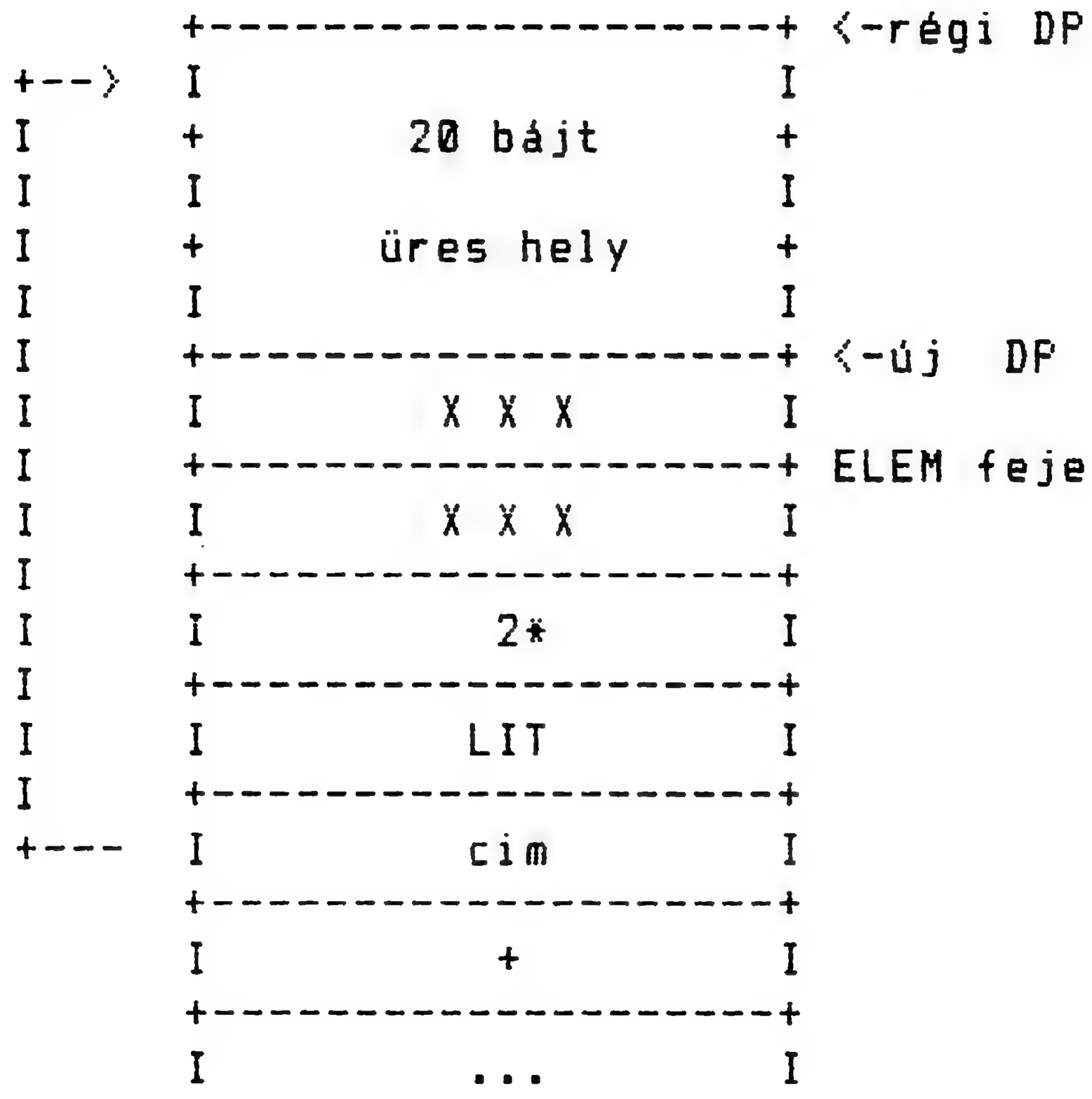
Tekintsük a LITERAL egy másik alkalmazását! Szeretnénk egy olyan tömböt létrehozni, amely egymás után tiz darab 16 bites számot tud tárolni. Az egyes elemeket valamilyen

egyszerű módon akarjuk elérni:

```
HERE DUP 20 + DP !
: ELEM 2* LITERAL [ 0 ] + ; DROP
```

A működés a következő: a HERE a veremre teszi a szótár következő üres bájtjának cimét. Ehhez hozzáadunk huszat, mivel a tiz darab 16 bites szám húsz bájtot foglal el. Az eredményt beirjuk a DP-be. Ezzel tiz memóriaszónyi üres hely keletkezett a szótárban. A DP régi értéke (vagyis az üres helyek kezdőcime) még ott van a vermen a DUP miatt. (Későbbi ismereteket (5.1.1. pont) felhasználva az első sor egyszerűbben HERE 20 ALLOT formában is irható).

Ezután jön az ELEM szó definiciója. Előbb bekerül a szótárba a "2*", azután a LITERAL beirja a LIT-et és a cimet. Vegyük észre, hogy a LITERAL által a szótárban elhelyezett szám most a szódefinición kivülről jött. Ez nem baj, de az viszont igen, hogy a FORTH elvárja a verem mélységének azonosságát a : és a ; végrehajtásakor. Ha a ; elérésekor a verem mélysége különbözik attól, ami a :-nál volt, a FORTH hibajelzést ad. Mivel a LITERAL egy számot levett a veremről, helyette egy másikat kell odatenni. Ezért szükséges a [0], amely a verem eredeti mélységét visszaállitja. A + szótárba forditása után a szódefinició befejeződik. A DROP-pal most már megszabadulhatunk a felesleges nullától. A szótár tehát a 3.11. ábra szerint alakul.



3.11. ábra
20 üres bájt a szótárban az ELEM előtt

Hogyan kaphatjuk meg az i-edik 16 bites szó cimét? A 0. cime cim+0, az elsőé cim+2 a másodiké cim+4 és igy tovább. Az i-edik elem cime általánosan cim+2*i. Vegyük észre, hogy az ELEM szó éppen ezt a számitást végzi el! Megszorozza kettővel az elem sorszámát, majd hozzáadja a kezdőcimet (ami a LIT cim miatt kerül a veremre, amit viszont a LITERAL tett a szótárba). Az ötödik elembe igy irhatunk, és belőle igy olvashatunk:

```
93 5 ELEM ! ok
```

5 ELEM @ . 93 ok

Bár célunkat, a tömb létrehozását elértük, különösen boldogok nem lehetünk. Igaz, a bemutatott módszer nem túl bonyolult, viszont nem elég rugalmas. Az ötödik fejezetben majd látni fogjuk, hogy a <BUILDS és DOES> szavak egy sokkal könnyebben kezelhető módszert biztositanak számunkra az adatszerkezet létrehozására.

Duplapontos számok forditására szolgál a **DLITERAL**. Működésére a LITERAL-nál elmondottak vonatkoznak.

3.6. Rendszervezérlő utasitások

3.6.1. ABORT, SP!, RP!

Mindenki követ el hibát, igy előfordul, hogy programozói hiba miatt a rendszer nem kivánt állapotba kerül. Tekintsük például a következő szót, amelynek feladata az első száz természetes szám kinyomtatása:

```
: KIIRJA 100 0 DO I LOOP ;
```

Ha ezt kipróbáljuk, a következőt kapjuk:

KIIRJA ok

Látjuk, hogy a KIIRJA nem irja ki a nullától 99-ig terjedő számokat, mert kifelejtettük az I után a . szót. Viszont a vermen ott található figyelmetlenségünk eredménye, a száz darab szám. Hogy az ilyen és ehhez hasonló hibákat helyrehozzuk, használjuk az **ABORT** (elvetélni) szót! Az ABORT törli mind az adat-, mind a visszatérési vermet és a FORTH interpreter állapotba kerül.

A verem törlését egyébként más módon is elérhetjük. Az **SP!** szó az adat-, az **RP!** a visszatérési vermet törli. Ezeket az ABORT is végrehajtja. Egy másik módszer: irjunk le egy tetszőleges ismeretlen szót, például azt, hogy QWERTY. Ilyen nevű szó a FORTH-ban nincs, és általában mi is ritkán definiálunk ilyet, mert a QWERTY név nem valami kifejező. Igaz, hogy hibajelzést fogunk kapni, de mindkét verem törlődni fog.

BEGIN...AGAIN ciklus megszakitását is elérhetjük az ABORT segitségével, de akkor azért esetleg "jobban" kiléptünk, mint akartunk, mert a vezérlés nem folytatódik az AGAIN után, hanem a rendszer alapállapotba kerül, és új felhasználói parancsra vár. Gyakran alkalmazzuk a következő szerkezetet:

```
... hibafeltétel IF ." hibaüzenet " ABORT ENDIF ...
```

Ez is egy módja annak, hogy például egy szó végrehajtása rossz bemeneti adatok esetén hibaüzenettel félbeszakadjon.

3.6.2. QUIT

A **QUIT** (= távozni, elhagyni) az ABORT-hoz hasonló szó. Mind interpreter, mind compiler üzemmódban használható. A QUIT törli a visszatérési vermet (az adatvermet változatlanul hagyja!) és hatására a FORTH interpreter módba kapcsol. Fontos tulajdonsága, hogy ha a QUIT végrehajtódik, akkor a FORTH nem irja ki szokásos "ok" üzenetét. Ezért például táblázatok kiiratásánál használhatjuk, megakadályozandó az "ok" kiirását a táblázat végén. Nézzük meg ezt a gyakorlatban a 3.3.2.2. pontban definiált EGYSZEREGY szó segitségével:

```
. EGYSZEREGY 11 1 DO CR
                    11 1 DO J I * 4 U.R LOOP
                    LOOP ; ok
EGYSZEREGY QUIT
```

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Az "ok" most nem jelent meg.

Ezenkivül használhatjuk a QUIT-et ciklusok rendellenes megszakitására, szavakból való kilépésre. Az ABORT-ot is gyakran a QUIT-tel helyettesitjük, ha az adatvermet nem akarjuk törölni.

3.6.3. COLD, WARM

Ez a két rendszervezérlő szó nem található meg minden FORTH implementációban. Az Olvasó könnyen ellenőrizheti, hogy saját FORTH rendszere tartalmazza-e a fenti szavakat: le kell irni őket, és a FORTH majd jelzi, ha nem ismeri.

A **COLD** (= hideg) a rendszer újrainditását szolgálja. Hatására törlődik a teljes felhasználói szótár, a rendszer-változók felveszik kezdeti értékeiket, azaz a rendszer olyan állapotba kerül, mintha éppen akkor inditottuk volna.

A **WARM** (= meleg) a FORTH melegstart. A rendszerváltozók kezdeti értékükre állitódnak, a vermek

törlődnek, de a felhasználói szótár érintetlen marad (ez az egyetlen különbség a COLD-hoz képest).

3.7. Feladatok

Az alábbi feladatok megoldásait az F1. függelékben adjuk meg.

3.7.1. Valósitsa meg a PICK szót!

3.7.2. Valósitsa meg a ROLL szót!

3.7.3. Oldja meg a 2.10. pontban található feladatokat olyan formában, hogy a megoldásként szereplő utasitássorozatokat irja meg szódefinicióként! Például:

```
: 2PICK ( utasitások ) ;
```

3.7.4. Készitsen olyan szót, amely beolvas egy karakterfüzért, és azt 32 bites előjeles számmá alakitja át!

3.7.5. Az előző feladat eredményeképpen keletkezett szó felhasználásával olvasson be n darab számot (n-et előzőleg helyezze a verem tetejére)! Irjon olyan szót, amely ezen számok átlagát képezi és az eredményt kiirja az alábbi formában:

```
a(z) n darab szám átlaga:xxxxxxxxxxxx
```

(A szám 12 karakter mezőszélességben iródjon ki!)

3.7.6. Irjon hatványozó szót! A két bemenő paraméter a következő:
Az alap 16 bites előjeles szám, a kitevő előjel nélküli 16 bites egész. Az eredmény 16 bites előjeles egész legyen. Túlcsordulás esetén jelezzen hibát!

3.7.7. Irja meg a SMUDGE szó segitségével, az euklideszi rekurziv formula felhasználásával a két szám legnagyobb közös osztóját kiszámitó szót! Az euklideszi algoritmus a következőképpen fest:

```
lko(n1,n2) = n1                      , ha n2=0
lko(n1,n2) = n2                      , ha n1=0
lko(n1,n2) = lko(n2,n1)              , ha n2>n1
lko(n1,n2) = lko(n2,n1 mod n2) egyébként
```

3.7.8. Irja meg az előbbi szót a MYSELF szó felhasználásával!

3.7.9. Irjon szót, amelynek segitségével az alábbi módon lehet összeadni:

```
5 MEG 8<R> = 13 ok
```

Adjon két megoldást a TEXT, ill. a WORD szó felhasználásával!

3.7.10. Irjon olyan szót, amely 1 és 2000 közötti egész számokat római számokká alakit, és az eredményt ki is irja! Példa:

```
1 ROMAI I ok
1985 ROMAI MCMLXXXV ok
```

4. A helyes programozási stilus FORTH-ban

A FORTH ellenzőinek legfőbb érve: a forrásszövegek nehezen olvashatóak. Ez az érv azonban csak azok számára elfogadható, akik a nyelvet kevéssé ismerik, vagy ismerik ugyan de olyan listát kaptak a kezükbe, amelynek készitője nem tartotta fontosnak, hogy betartsa a következő formai követelményeket:

- egyszerűség,
- inkább 3 rövid, mint egy hosszú szó;
- jó névválasztás (szó, változó, konstans azonositók),
- külalak (tabuláció),
- sok, érthető kommentár.

Ezek a helyes FORTH programozási stilus formai elemei. Mivel azonban minden elmélet csak annyit ér, amennyi használható belőle, lássunk egy példát. Az Eratoszthenész szitája nevű, jól ismert, primszámok előállitására szolgáló algoritmust valósitjuk meg, két példányban. A különbség szemmel látható.

```
: ERATOSTHENES PAD 128 1 FILL 128 2 DO I DUP PAD + C@ 1 =
   IF DUP 2 * 128 SWAP DO 0 I PAD + C! DUP +LOOP DROP ELSE
   DROP ENDIF LOOP 128 2 DO PAD I + C@ 1 = IF I . ENDIF
   LOOP ;
```

Ez egy teljesen érthetetlen változat, most próbáljuk másként:

```
2   CONSTANT ALSO.HATAR
128 CONSTANT FELSO.HATAR
: HATAR1 PAD FELSO.HATAR + ;
: HATAR2 PAD  ALSO.HATAR + ;
: NULLAZAS                  ( j n --- )
        PAD +               ( j n --- j PAD+n )
        HATAR1 SWAP         ( j PAD+n h1 --- j h1 PAD+n )
        DO
          0 I               ( j --- j 0 i )
          C!                ( j 0 i --- j )
          DUP               ( j --- j j )
        +LOOP               ( j j --- j )
        DROP                ( j --- )
;
: ELAGAZAS                  ( i --- )
        DUP                 ( i --- i i )
        PAD +               ( i i --- i PAD+i )
        C@                  ( i PAD+i --- i n )
        1 =                 ( i n --- i f )
        IF                  ( ha nem volt kinullázva )
          DUP               ( i --- i i )
          2 *               ( i i --- i 2*i )
          NULLAZAS          ( i 2*i --- )
        ELSE
          DROP              ( i --- )
        ENDIF
;
: SZITALAS                  ( --- )
        FELSO.HATAR ALSO.HATAR ( --- fh ah )
        DO
          I ELAGAZAS        ( primszámok keresése )
        LOOP
;
```

```
: KIIRATAS
        FELSO.HATAR ALSO.HATAR    ( --- fh ah )
        DO
          I PAD + C@      ( --- n )
          1 =             ( n --- f )
          IF              ( ha primszám )
            I .
          ENDIF
        LOOP
;
: INICIALIZALAS           ( --- )
        PAD FELSO.HATAR 1
        FILL              ( PAD fh 1 --- )
;
: ERATOSTHENES
        INICIALIZALAS
        SZITALAS
        KIIRATAS ;
```

A 4.2.3. pontban sorravesszük a program megvalósitásának lépéseit.

A helyes FORTH programozási stilus másik fő eleme a FORTH rendszer hatékony felhasználása. Az 1.2. szakaszban megismertük a FORTH rendszer által sugallt programfejlesztési folyamatot. Ehhez szeretnénk hozzáfűzni néhány gyakorlati útmutatást. A papiron megirt programot szavanként gépeljük be, majd azonnal próbáljuk is ki! Addig ne haladjunk tovább, amig a működés nem helyes! Igy maximálisan kihasználhatjuk a FORTH rendszer által kinált előnyöket.

4.1. A "bottom-up" programozási módszer

A programozás irodalmában alapvetően kétféle program-szintetizálási metódust különböztetünk meg. Ezek egyike a "bottom-up" módszer (legismertebb képviselője az APL programozási nyelv).

Amikor egy problémát számitógéppel akarunk megoldani, úgy két fix pont adott: a rendelkezésre álló gép utasitáskészlete (itt most nincs különbség a magas és alacsony szint között), és a megoldandó probléma. A tárgyalt programozási módszer lényege, hogy a probléma megoldását a gép felől kezdi meg, és attól egyre inkább elvonatkoztatva halad a feladat irányába. Az eljárást olyan LEGO játékhoz hasonlithatnánk, amelyben a kockáknak a gép utasitásai felelnek meg, és ezekből kell felépitenünk egy bonyolult épitményt.

4.2. Moduláris programozás

A következő alpontok a moduláris programozást elméleti oldairól próbálják megközeliteni és némi betekintést próbálnak nyújtani a korszerű és elméletileg megalapozott programozási módszerekbe. A 4.2.3. pontban egy adott példára fogjuk alkalmazni az ismertetett elveket.

4.2.1. Logikailag különálló programegységek

Valódi alkalmazásoknál gyakran előfordulhat, hogy akár 500-1000 programsorral oldjuk meg a feladatot. Azonban egy

ekkora program fogalmilag nem nevezhető nagynak. Ha a hierarchikusan egymás alatt álló modulokat (alkalmasint szavakat) megfelelően terveztük meg, azaz egyik sem tud a szükségesnél többet a másikról, akkor nagy a valószinűsége, hogy a program áttekinthető lesz. A későbbi módositások is könnyebbé válnak azáltal, hogy a modulok a lehető legnagyobb mértékben el vannak szigetelve egymástól, a szükséges kapcsolatok pedig jól áttekinthetőek. A sokszor használt konstansok alkalmasan választott jelölésével szintén sok felesleges mechanikus munkától (editorozás) szabadulhatunk meg.

4.2.2. A "top-down" programozási elv

A "top-down" elv (másképpen lépésenkénti finomitás = stepwise refinement) a 4.1. szakaszban tárgyalt módszer komplementere, vagyis a probléma és a gép között "tátongó rést" a probléma felől igyekszik áthidalni. Lényege a következőkben foglalható össze: először a program algoritmusának leglényegesebb részeit irjuk meg, majd a program többlépéses bővitésével, finomitásával egyre részletesebbé alakitjuk programunkat, amig elérünk az adott programozási környezet által megkövetelt részletességű és pontosságú forrásszöveghez.

A "top-down" elv és a struktúrált programozás fogalma nagyon erősen kapcsolódik egymáshoz. Manapság mindinkább a struktúrált programozás korában élünk. Hogy megértsük mit is jelent ez, emlékezzünk vissza néhány, a 3.3. szakaszban tárgyalt alapfogalomra: tevékenység, választás, ismétlés. Nos ha a program gráfja ezekkel, és csak ezekkel a fogalmakkal leirható, akkor beszélünk struktúrált programról. (Gráf: egy csúcspontokból és az azokat összekötő vonalakból, élekből álló alakzat.)

A gyakorlati programozásban ennek a haszna két oldalról jelentkezik, egyrészt az úgynevezett struktúrált programozási nyelveken irott programok jól áttekinthetőek, másrészt a jelenleg még gyerekcipőben járó programhelyesség-bizonyitó eljárások csak ilyen programokra alkalmazhatóak.

4.2.3. Alkalmazási példa

Mivel véleményünk szerint sem a tisztán top-down (l. a 4.2.2. pontot) módszerrel, sem a tisztán bottom-up (l. a 4.1. szakaszt) módon való programozás nem célravezető, javasolt módszerünk a tervezési fázisban top-down, az implementálási fázisban bottom-up programozási elv követése.

Kövessük végig az ismertetett elvek használatát a 4. fejezet elején bemutatott ERATOSTHENES nevű programon (természetesen a szebbik változatra gondolunk)! Az algoritmust ismertnek tételezzük fel.

A tervezés első lépése az adatok tárolási módjának eldöntése. Jelen esetben egy, a PAD-től kezdődő memóriaterületen helyezzük el a tömb elemeit. Igy a kezdőcim és az elem sorszáma alapján maga az elem könnyen elérhető.

A program minden bizonnyal a kezdeti értékek beállitásával indul. Nevezzük ezt a tevékenységet INICIALIZALAS-nak. Ezután a következő lépés nyilván a szitálási algoritmus végrehajtása, nevezzük ezt SZITALAS-nak. Végül az eredményeket ki kell iratnunk, hivjuk ezt a részt KIIRATAS-nak. Vegyük észre, hogy eddig az egyes részeket alkotó konkrét tevékenységekkel nem foglalkoztunk, csak a főbb részeket különitettük el. Programunk eddig az alábbi részekből áll:

ERATOSTHENES => INICIALIZALAS SZITALAS KIIRATAS

Az INICIALIZALAS esetünkben a munkaterület 1-esekkel való feltöltését jelenti, ezért több finomitási lépésre nincs is szükségünk. Ezért ezt rögtön megvalósithatjuk az adott nyelv eszközeivel.

Az algoritmus fő része a SZITALAS. Legfelsőbb szintje egy ciklus, amelyben megvizsgáljuk, hogy a szám lehet-e primszám. Ezt a vizsgálatot egy külön tevékenységre bizzuk, nevezzük ezt például ELAGAZAS-nak.

SZITALAS => ciklus ELAGAZAS ciklusvég

Finomitsuk tovább az ELAGAZAS-t. Arra van szükségünk, hogy azoknak a tömbelemeknek a többszöröseit, amelyek nincsenek kinullázva, ki kell nulláznunk. Ezt a tevékenységet NULLAZAS-nak neveztük el.

ELAGAZAS => ha elem=1, akkor NULLAZAS

A NULLAZAS olyan tevékenység, amelynek további finomitása már nem szükséges, a ciklust azonnal megirhatjuk FORTH-ban.

A KIIRATAS kiirja az olyan elemek sorszámát, amelyek értéke 1 (azaz primszámok). Ennek további finomitása szintén szükségtelen, hiszen ezt is azonnal megirhatjuk.

Láthatjuk tehát, hogy a top-down elv alkalmazásával a feladatot olyan részekre bonthatjuk, amely részek bottom-up módszerrel már könnyen és hatékonyan elkészithetők.

4.3. A FORTH kiterjeszthetősége

A FORTH egyik legfontosabb sajátossága, hogy utasitáskészlete tetszőlegesen bővithető, igy a nyelv egy-egy adott feladat igényeinek megfelelően kiterjeszthető. A kiterjesztésnek három fontosabb iránya lehet:

a) Assembly

Amennyiben összehasonlitjuk a magas és alacsony szintű programozási nyelveket úgy azt tapasztaljuk, hogy a magas szintű nyelvek inkább elszigetelnek a géptől (erre jó okuk a hordozhatóság biztositása), mig az alacsony szintű assembly nyelvek erősen függenek az elkészités bázisául szolgáló rendszer sajátosságaitól (hardver, szoftver, operációs rendszer). A FORTH megalkotói igyekeztek párositani az assembly nyelv gyorsaságát a magas szintű nyelvek előnyeivel. Ennek következménye a beépitett assembler létrehozása.

b) Magasszintű nyelvi eszközök létrehozása

A kiterjesztés másik iránya új magasszintű nyelvi eszközökhöz vezet. Például a FORTH nem tartalmazza a CASE..OF..ENDOF..ENDCASE vezérlési szerkezetet. Példánk rámutat, hogy szükség esetén ilyen létrehozható. Itt ne törekedjünk a közölt program megértésére, a benne előforduló ismeretlen szavakat a ó. fejezetben részletezzük. Csak arra a tényre szeretnénk rávilágitani, hogy egy adott feladathoz tökéletesen illeszkedő, akár más programozási nyelvben elő sem forduló vezérlési szerkezet létrehozása is lehetséges.

```
: CASE ?COMP CSP @ !CSP 4 ; IMMEDIATE
: OF 4 ?PAIRS COMPILE OVER COMPILE = COMPILE
   0BRANCH HERE 0 , COMPILE DROP 5 ; IMMEDIATE
: ENDOF 5 ?PAIRS COMPILE BRANCH HERE 0 ,
      SWAP 2 [COMPILE] ENDIF 4 ; IMMEDIATE
: ENDCASE 4 ?PAIRS COMPILE DROP
         BEGIN SP@ CSP @ = 0=
         WHILE 2 [COMPILE] ENDIF
         REPEAT CSP ! ; IMMEDIATE
```

Szintaxisa a következő:

```
CASE ( n --- )
     n1 OF szósor1 ENDOF ( ha n=n1 )
     n2 OF szósor2 ENDOF ( ha n=n2 )
                  .
                  .
     nn OF szósorn ENDOF ( ha n=nn )
        szósor.egyéb      ( ha n<>n1,n2...nn )
   ENDCASE
```

c) Adatstruktúrák létrehozása

Harmadik fontos területe a nyelvi eszközök bővitésének az adatstruktúrák differenciálása, új adatstruktúrák létrehozása. Ezzel részletesen az 5. fejezetben foglalkozunk.

5. Adatkezelés

A programok két összetevőből állnak: algoritmusokból, amelyek megvalósitásával a 3. fejezet foglalkozott, és adatstruktúrákból. Az adatszerkezetek lehetnek egyszerű adattipusok (l. 2.2. szakaszt). valamint összetett adatobjektumok (vektor, tömb, rekord stb.). A következőkben ismertetjük, hogy az összetett adatszerkezetek létrehozására milyen nyelvi eszközöket kinál a FORTH.

5.1. Helykijelölés

A ma használatos számitógépek többnyire a Neumann-elven működnek. Ez többek között azt jelenti, hogy a program és az adatok egyaránt a számitógép operativ tárában helyezkednek el. A jelen esetben ez azzal a következménnyel jár, hogy ha új adatstruktúrát kivánunk létrehozni, úgy nekünk kell biztositani az adatobjektum elemeinek elhelyezésére szolgáló helyet. Az alább ismertetendő utasitások a felhasználói szótár utáni memóriarészben foglalnak le helyet. A 3.2.1.2. pontban részletezett DP rendszerváltozó ismeretét feltételezzük.

5.1.1. ALLOT

Az **ALLOT** utasitás (n ---) n darab bájtot lefoglal az operativ tár felhasználói szótár utáni részében. Végrehajtása után a DP rendszerváltozó a lefoglalt terület utáni első szabad bájtra mutat.

```
n ALLOT = n DP +!
```

Például egy 80 karakter hosszú karakterfüzér számára igy foglalhatunk helyet:

80 ALLOT ok

5.1.2. , és C,

A , utasitás (n ---) az adatvermen található 16 bites számot elhelyezi a DP által mutatott cimre, egyszersmind a DP értékét kettővel megnöveli, igy az a következő szabad helyre mutat.

n , = n HERE 2 ALLOT !

A C, utasitás (n ---) a vermen található 16 bites szám alsó 8 bitjét helyezi a felhasználói szótár végére. A DP értéke eggyel nő.

n C, = n HERE 1 ALLOT C!

5.2. Adatszerkezetek létrehozása

Amint a bevezetőben emlitettük, az adatszerkezetek kétfélék lehetnek, egyszerűek és összetettek. A FORTH nyelvben rendelkezésre álló egyszerű adattipusokat a 2. fejezetben már megismertük (egész, duplapontos egész). Bizonyos összetett adatobjektumok is ismertek már előttünk (karakterfüzér). Most a bonyolultabb összetételi eszközöket tárgyaljuk.

5.2.1. CREATE, <BUILDS..DOES>

Elérkeztünk a FORTH nyelv fogalmilag talán legnehezebben érthető részéhez. A könnyebb megértés végett először tisztáznunk kell a futásidő és forditásidő fogalmát.

5.2. Adatszerkezetek létrehozása

A FORTH rendszerekben fordításidő az az idő amelyben a FORTH leforditja a begépelt definiciókat. Futásidőről pedig akkor beszélünk, amikor egy szó végrehajtása történik. Ennek szemléltetésére tekintsünk egy példát:

```
: FORDIDO ." forditasido" ; IMMEDIATE ok
: FUTIDO FORDIDO ." futasido" ; forditasido ok
FUTIDO futasido ok
```

A FORTH-ban léteznek olyan szavak - generáló szavak -, amelyek működésük során új szavakat hoznak létre (:, CONSTANT, VARIABLE). Az itt tárgyalt utasitások segitségével ilyen generáló szavakat hozhatunk létre. A **CREATE** utasitásnál két esetet kell megkülönböztetnünk. A fig-FORTH-ban a CREATE mindössze egy szótári bejegyzést hoz létre:

```
CREATE szónév
```

A FORTH-79 CREATE-je azonos a fig-FORTH <BUILDS utasitásával, azzal az eltéréssel, hogy a SMUDGE bit állitása különbözik. A <BUILDS által létrehozott szó rögtön érvényes is lesz. Ahhoz azonban, hogy meg is hivhassuk ezt a szót, algoritmust kell hozzákapcsolni, amit éppen a **<BUILDS..DOES>** szerkezettel lehet megtenni.

A továbbiakban a **<BUILDS..DOES>** utasitáspárt részletezzük:

Először hozzuk létre a generáló szót:

```
: GENERALOSZO <BUILDS generáló szósor
              DOES>   végrehajtási szósor ;
```

Ezt felhasználva hozzunk létre új szavakat:

```
GENERALOSZO UJSZO1   ( UJSZO1 létrehozása )
GENERALOSZO UJSZO2   ( UJSZO2 létrehozása )
        .            ( további szavak létrehozása )
        .
        .
```

Az UJSZO1, UJSZO2, ... létrehozása során a GENERALOSZO "generáló szósor" része hajtódik végre (fordításidő). A generált szavak (UJSZO1, UJSZO2, ...) használata esetén azok szótári cime felkerül a veremre, majd a vezérlés a "végrehajtási szósor"-ra adódik át (futásidő).

Ezt a nem túl könnyen emészthető gondolatsort egy példával próbáljuk meg érthetőbbé tenni. Hozzunk létre egy KONSTANS szót, amelynek funkciója megegyezik a 2.3.2. pontban leirt CONSTANT szóéval, vagyis konstansok generálását oldja meg.

```
( generáló szó )
: KONSTANS ( n --- )
           <BUILDS    ( fordításidő: n --- )
           ,          ( HERE-re leteszi n-et )
           DOES>      ( futásidő: --- cim )
           @          ( cim --- n )
;

( konstans generálása )
15 KONSTANS CITROM    ( fordításidő )
```

Elemezzük CITROM fordításidejét: végrehajtódik a , utasitás, azaz a 15 bekerül a DP által mutatott helyre (a futásidőben majd ez a cim kerül a veremre!!), majd DP megnövekszik kettővel.

```
CITROM . 15 ok          ( futásidő )
```

Most vegyük szemügyre a futásidejű működést: a verem tetején először megjelenik a CITROM szótárbeli cime, majd a KONSTANS szó DOES> utáni részére kerül a vezérlés, vagyis erről a cimről a @ utasitás felteszi a veremre a 15 számot. Ezt azután a ponttal kiiratjuk a képernyőre. A bonyolultabb alkalmazási példák előtt még lássuk a VARIABLE szónak megfelelő VALTOZO-t (n VALTOZO szónév):

```
: VALTOZO <BUILDS 2 ALLOT DOES> ;
```

Ennek megértését az Olvasóra bizzuk.

5.2.2. Példák adatszerkezetek létrehozására

5.2.2.1. n-elemű vektor létrehozása

```
: VEKTOR ( n --- )
         <BUILDS      ( n --- )
           2 * ALLOT  ( n*2 bájt lefoglalása )
         DOES>        ( n1 --- n1 cim )
           SWAP       ( cim n1 )
           1-         ( cim n1-1 )
           2 *        ( cim 2*[n1-1] )
           +          ( cim+2*[n1-1] = n1. elem cime )
;
```

ok

Most hozzunk létre egy 10 elemű vektort SZAMSOR néven:

```
10 VEKTOR SZAMSOR
```

ok

Azt már tudjuk, hogy a SZAMSOR meghivásakor előbb a SZAMSOR nevű szó cime a veremre kerül, majd végrehajtódik a generáló szó (jelen esetben a VEKTOR) DOES> utáni része. Ha például a harmadik elembe be akarunk tenni 66-ot, akkor a következőt kell leirnunk:

```
66 3 SZAMSOR ! ok
```

Ennek hatására a következő történik:

```
66 3 ( --- 66 3 )
SZAMSOR
                ( 66 3 --- 66 3 cim )
                ( SWAP 66 3 cim --- 66 cim 3 )
                ( 1-   66 cim 3 --- 66 cim 2 )
                ( 2 *  66 cim 2 --- 66 cim 4 )
                ( +    66 cim 4 --- 66 cim+4 )
!     ( 66 cim+4 --- )
```

A példában vastagitva jelölt részek azok, amelyeket mi irunk le. A többi a DOES> utáni, a SZAMSOR leirásakor végrehajtódó útasitássorozat.

Adjunk hozzá a 3. eleméhez 5-öt:

```
5 3 SZAMSOR +! ok
```

Irassuk ki a tartalmát:

```
3 SZAMSOR ? 71 ok
```

5.2.2.2. Karakterfüzér-konstans létrehozása

Példánk forditásidőben a TEXT utasitást felhasználva beolvas egy karakterfüzért az bemeneti pufferből a PAD-re, ezt eltárolja DP-től, majd ALLOT-tal helyet foglal le neki. A

füzér hossza DP-re, a füzér DP+1-re kerül. (l. a 2.6. ábrát). Futásidőben a DOES> utáni rész ezt a füzért kiirja.

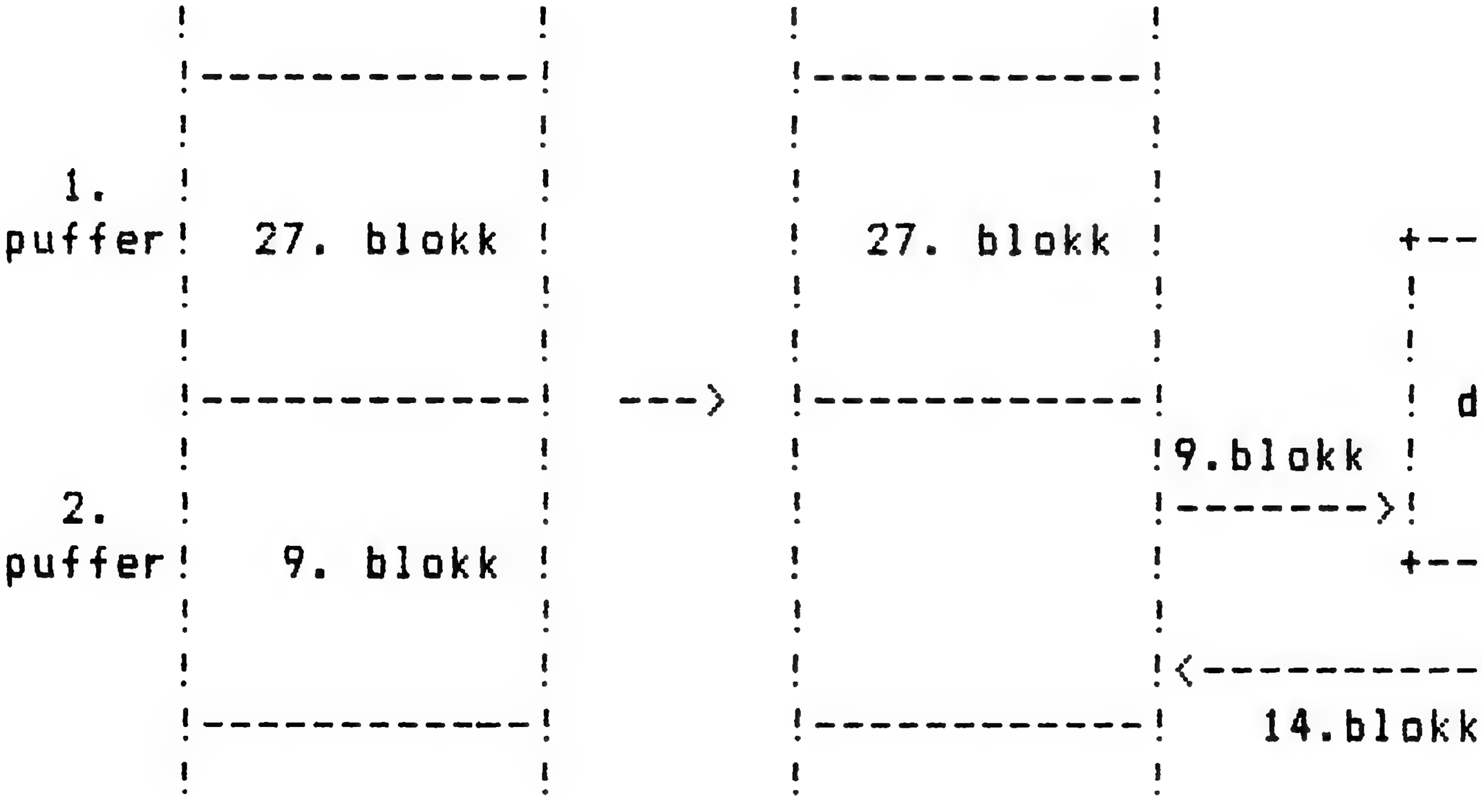
```
: $CONSTANT
          <BUILDS
            13 TEXT    ( CR-ig beolvassa a füzért
                         PAD-re az bemeneti pufferből )
            PAD C@     ( hossz )
            1+         ( hossz+1 )
            DUP        ( hossz+1 hossz+1 )
            PAD        ( hossz+1 hossz+1 PAD )
            HERE       ( hossz+1 hossz+1 PAD DP )
            ROT        ( hossz+1 PAD DP hossz+1 )
            CMOVE      ( hossz+1 )
                       ( hossz+1 bájt másolása
                         PAD-ről DP-re )
            ALLOT      ( DP növelése hossz+1-gyel )
          DOES>        ( cim )
            COUNT      ( cim --- cim+1 hossz )
                       TYPE
;  ok

$CONSTANT WAGNER1 NA MI UJSAG<R> ok
$CONSTANT WAGNER2  WAGNER UR?<R> ok
WAGNER1 WAGNER2 NA MI UJSAG WAGNER UR? ok
```

5.3. Virtuális memóriakezelés

Murphy egyik, nem a legpozitivabb életérzést sugalló törvénye szerint "minden program a rendelkezésre álló memória határáig nő." Ennek a valóságmagra épülő felismerésnek következményei már régóta foglalkoztatják a számitógépes szakembereket. A számukra legérdekesebb megoldás lehetővé teszi a nagy kapacitású háttértárak (többnyire mágneslemez =

diszk) operativ tárként való felhasználását. Ezt a módszert nevezik virtuális (= látszólagos) memóriakezelésnek. Lényege a következő: az operativ tárban kijelölnek néhány meghatározott nagyságú területet (továbbiakban blokkpufferek). A mágneslemez egységeken az információtárolás fizikai blokkokban történik, ezek közül azonban egyszerre többet használnak egy egységként az adatforgalom során. Ezeket a logikailag összefüggő blokkokat logikai blokknak, a továbbiakban blokknak nevezzük. A blokkpufferek mérete megegyezik e blokkok méretével (általában 1..4 kbájt). Az igazi érdekesség az adatátvitelt ellenőrző programban van. Ugyanis ez a program megjelöli a behozott blokkokat, és ha már az összes blokkpuffer betelt, de újabb blokk behozására van szükség, akkor azt a blokkot, amelyre legrégebben volt hivatkozás, visszairja a mágneslemez egységre. Kövessük az elmondottakat az 5.1. ábrán!



5.1. ábra
Virtuális memóriakezelés

Beolvassuk a 27. és 9. blokkot, de csak a 27-est dolgozzuk fel. Ezért ha be akarjuk olvasni a 14-est, akkor a 9-es visszakerül a mágneslemez egységre, és helyét elfoglalja a 14-es.

A fenti módszert a FORTH rendszerekben is alkalmazzák, a mikroszámitógépek többnyire 64 kbájtos memóriáját igy több száz kilobájtra bővitik.

A FORTH rendszerekben a blokkpufferek hosszát a **B/BUF** konstans adja meg, értéke általában 1024 bájt (= 1 kbájt). Minden pufferterülethez hozzá van rendelve egy úgynevezett Blokk Állapot Cella (= Block Status Cell), amely tartalmazza az adott pufferben levő blokk sorszámát. Az utoljára hivatkozott blokkpufferének cime a **PREV** változóba kerül. Ezt a blokkot a továbbiakban aktuális blokknak nevezzük.

5.3.1. Pufferek kezelése

5.3.1.1. BLOCK

A **BLOCK** utasitás (u --- cim) egy adott blokk betöltését végzi a mágneslemez egységről egy blokkpufferbe, ha az még nincs a memóriában. Bemenő adata egy 16 bites előjel nélküli szám, amely a beolvasandó blokk sorszáma. A végrehajtás során a FORTH megvizsgálja a blokkpuffereket. Amennyiben nincs már üres hely, úgy a legrégebben használt blokkot visszairja a mágneslemez egységre, ha az módositva volt. A lefutás után a vermen megtalálható a behozott blokkot tartalmazó blokkpuffer kezdőcime. A továbbiakban ez lesz az aktuális blokk. Egy blokkot a képernyőre listáz a következő utasitássorozat:

```
u BLOCK B/BUF TYPE
```

5.3.1.2. BUFFER

A **BUFFER** utasitás (u --- cim) felszabadit egy blokkpuffert a verem tetején megadott sorszámú blokk számára. A blokkot nem hozza be a háttértárról. A blokkpúffer kezdőcimét a veremre helyezi. A továbbiakban ez lesz az aktuális blokk, ebbe helyezhetünk el adatokat.

5.3.1.3. UPDATE

Az **UPDATE** utasitás az aktuális blokkot tartalmazó puffer Blokk Állapot Celláját úgy állitja be, mintha azt módositottuk volna. Igy, ha szükség van az elfoglalt terület kiüritésére, akkor a puffer tartalma a mágneslemez egységre automatikusan visszairódik.

5.3.1.4. SAVE-BUFFERS, FLUSH

A FORTH-79 **SAVE-BUFFERS**, a fig-FORTH **FLUSH** néven ismeri ezt az utasitást. Az összes módositott puffert visszairja a mágneslemez egységre. A végrehajtást követően az összes Blokk Állapot Cella törlődik, igy ha munkánkat folytatni kivánjuk, a kimentett blokkokat újra vissza ,kell töltenünk.

5.3.1.5. EMPTY-BUFFERS

Az **EMPTY-BUFFERS** utasitás az összes puffer állapot bájtját átirja úgy mintha a pufferek üresek lennének. Az UPDATE-tel megjelölt blokkok sem iródnak vissza a diszkre. A pufferek aktuális tartalma az adott megvalósitástól függően megmarad, vagy nullákkal feltöltődik.

5.3.1.6. COPY

A **COPY** utasitás (u1 u2 ---) átmásolja az u1-edik blokk tartalmát az u2-edik blokkba (az u2-edik blokk eredeti

tartalma elvész).

5.3.1.7. WIPE

A WIPE az aktuális blokk tartalmát feltölti szóköz karakterekkel.

5.3.1.8. R/W

Ez a fig-FORTH blokk író-olvasó szava (cim u j ---). Ha j=0, akkor irunk a lemezre, egyébként olvasunk. Az u jelöli az átvinni kivánt blokk sorszámát, cim pedig az átvitelhez felhasználandó puffer kezdőcime.

5.3.2. A screen fogalma

A FORTH nyelvet eleve képernyővel és billentyűzettel rendelkező interaktiv kommunikációra képes számitógépre szánták. A screen (ejtsd: szkrin) angol szó, képernyőt, vetitőernyőt jelent. Mivel egy képernyőnyi információ jól áttekinthető, ez adhatta az ötletet, hogy a FORTH nagyobb programegységeit screennek nevezzék. Egy screen általában 16 sorból és soronként 64 karakterből áll, azaz összesen 1024 bájt = 1 kbájt hosszú. Az igazi zsenialitás abban rejlik, hogy a screenek kezelése a virtuális memóriakezelés szabályai szerint zajlik olyannyira, hogy az 5.3.1. pontban ismertett összes utasitás itt is felhasználható. Ezek fényében a screen nem más, mint egy olyan logikai blokk, amelynek FORTH forrásszöveg a tartalma. A screenek rendre egy-egy logikai blokknak felelnek meg. Például az 1. blokk tartalmazza az 1. screent. Azt, hogy egy screen hány fizikai blokkot tartalmaz, a B/SCR konstans adja meg. A legtisztább az eset, amikor B/SCR=1. Ekkor a blokk és a screen

szinonimák. De van olyan rendszer, ahol B/BUFF=128 és B/SCR=8, vagyis egy screen nyolc darab fizikai blokkból áll. Jobb ezért a blokk és a screen fogalmát pontosan elhatárolni egymástól.

5.3.3. Screenkezelés

5.3.3.1. LOAD

A LOAD utasitás (u ---) a vermen levő számot (amely 16 bites és előjel nélküli) egy screen azonositójaként értelmezi. Beolvassa az adott screent tartalmazó blokkokat. A forrásszöveget leforditja, az utasitásokat végrehajtja.

Itt kell megemlitenünk két fontos jelet. Az egyik a --->, ez azt jelzi az interpreternek, hogy a fordítást a következő screenben folytatni kell. A másik, a ;S jelzi, hogy a fordítás az adott screenben befejeződött.

5.3.3.2. LIST

A LIST (u ---) az u-adik screen tartalmát kilistázza a képernyőre az alábbi formában:

```
9 LIST
Screen # 9
  0 ( Veletlen szam generator, J. E. Rickenbacker )
  1 ( FORTH DIMENSIONS II/2 34. oldal               )
  2
  3 FORTH DEFINITIONS DECIMAL
  4
  5 0 VARIABLE SEED
  6
  7 : (RAND) SEED @ 259 * 3 + 32767 AND DUP SEED ! ;
  8
  9 : RANDOM (RAND) 32767 */ ;
 10
 11
 12
 13
 14
 15
```

5.2. ábra
Egy screen listája

A screen száma beiródik az SCR változóba, azaz ez lesz az aktuális screen.

Jegyezzük meg, hogy a kialakult szokások szerint a screen nulladik sorában a screen tartalmára utaló megjegyzés található!

5.3.3.3. INDEX

Az **INDEX** utasitás (u1 u2 ---) az u1. screentől az u2.-ig listázza a screenek nulladik sorát. Igy lehet gyorsan felderiteni a screenek tartalmát feltéve, hogy a LIST-nél emlitett konvenciót betartjuk.

5.3.3.4. TRIAD

A **TRIAD** (u ---) három screent listáz. Mivel általában ennyi fér el egy sornyomtató lapra, formatált listát nyerhetünk. Az u sorszámot csonkitja az u1<=u számra, amely az u-nál kisebb vagy azzal egyenlő legközelebbi hárommal osztható szám. A listázást ettől az u1. screentől kezdi meg, előzőleg azonban végrehajt egy lapdobást. A lap aljára a 15-ös üzenetet nyomtatja (l. 6.8. szakasz, MESSAGE).

5.3.3.5. (LINE), .LINE

A **(LINE)** utasitás (u1 u2 --- cim u) az u2. screen u1. sorának a memória pufferben levő kezdőcimét (cim) és hosszát (u) adja meg. Ha a screen nincs a pufferben, akkor betölti.

A **.LINE** (u1 u2 ---) az u2. screen u1. sorát listázza.

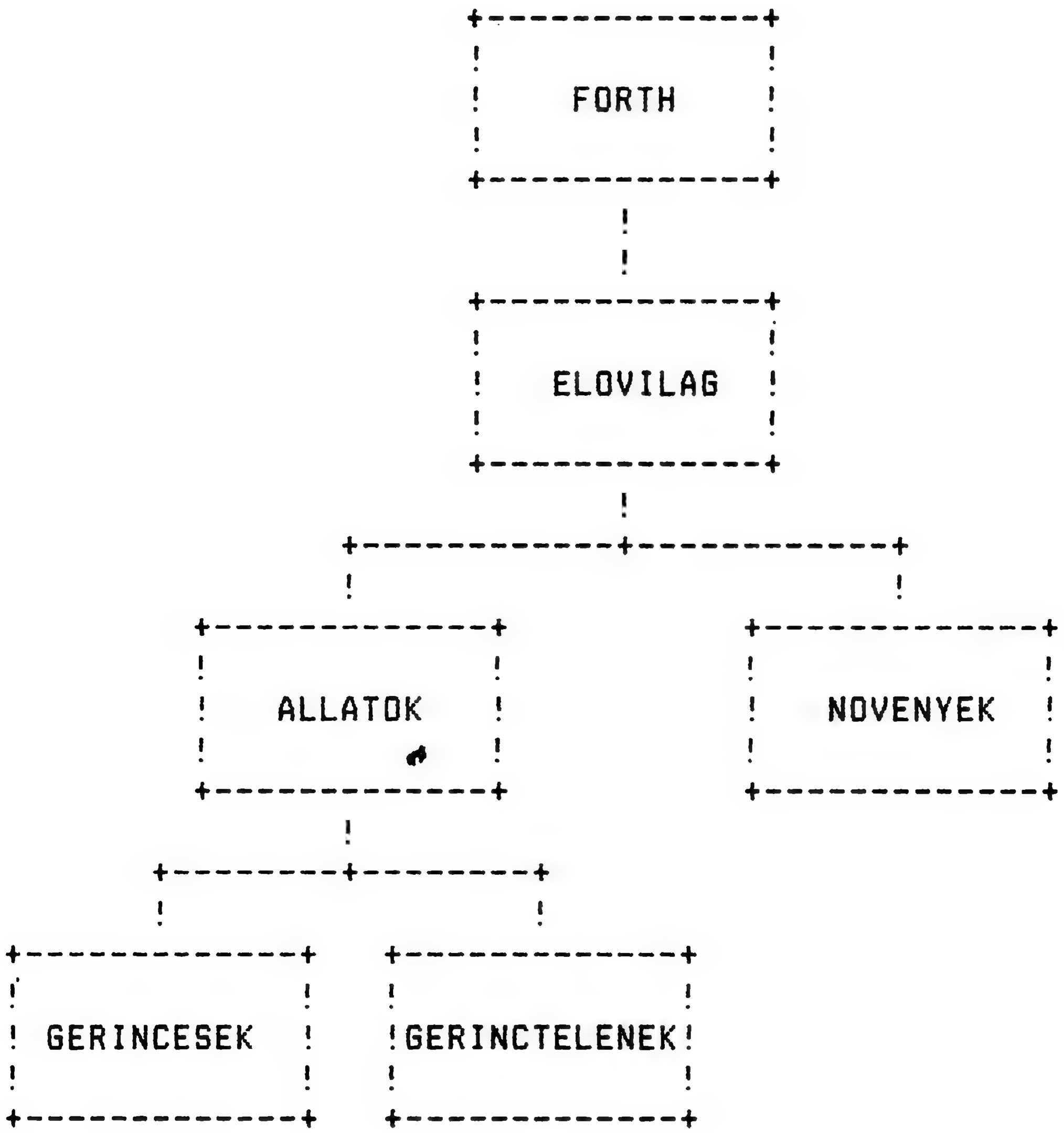
5.4. A felhasználói szótár használata

Léteznek olyan alkalmazások, amelyeknél több, egymástól logikailag különálló alkalmazói program egyidejű tárbeli jelenlétére van szükség (pl. műszervezérlés), de ezeket a programokat célszerű egymástól valami módon elkülöniteni. Már csak azért is fontos ez, hogy a gyakran használt változó-, szó- és konstansnevek ütközéseit elkerüljük. A felhasználói szótárakat ilyen célokra találták ki.

Az itteni szótárstruktúra kis mértékben eltér a 3.

fejezetben megismerttől. Az egy szótárba tartozó szavak továbbra is egy egységet alkotnak, de a szótárak egymástól jól elválaszthatók. A felhasználói szótárak a FORTH alapszótárától egy fa ágaihoz hasonlóan ágaznak el. Tételezzük fel, hogy a FORTH alapszótáron kivül van egy ELOVILAG, egy ALLATOK és egy NOVENYEK nevű szótárunk. Az ALLATOK szótáron belül létrehoztunk egy GERINCESEK és egy GERINCTELENEK nevű alszótárat is. Az 5.3. ábra bemutatja az elágazási struktúrát.

5.4. A felhasználói szótár használata



5.3. ábra
Felhasználói szótárak fája

5.4. A felhasználói szótár használata

A felhasználói szótárak alkalmazásánál két probléma merül fel: az egyik, hogy a további szódefiniciók amelyik szótárhoz tartozzanak, a másik, hogy az új definiciók fordítása közben a keresés amelyik szótárban történjék. A FORTH ezeknek a kérdéseknek az eldöntésére **CONTEXT** és **CURRENT** nevű rendszerváltozóban tárolt információt használja fel. A CONTEXT mindig annak a szótárnak a cimét tartalmazza, amelyben a keresés történik, a CURRENT-ben levő cim annak a szótárnak a cime, amelyhez az új definiciók tartozni fognak. A szótár cimén itt a szótárban utoljára elhelyezett szó cimét kell érteni, ezáltal egyszerűen lehetővé válik az utolsó elemtől visszafelé haladó keresés (l. még a 6. fejezetet).

Az alábbiakban a felhasználói szótárak kezelését szolgáló FORTH szavakat foglaljuk össze.

5.4.1. VOCABULARY

felhasználása: VOCABULARY szótárnév

Létrehozza az szótárnév nevű szótárt. Ez a "létrehozás" mindössze annyit jelent, hogy elteszi a "szótárnév" cimét. Igy, ha később begépeljük a szótárunk nevét, akkor a CONTEXT úgy módosul, hogy a keresés ezt követően a "szótárnév" szótárban fog történni.

A gyakorlatban a "szótárnev"-et közvetlen szóként definiáljuk, hogy a szótárak közötti váltás szódefinició közben is lehetséges legyen, azaz:

VOCABULARY szótárnév IMMEDIATE

5.4.2. DEFINITIONS

felhasználása: szótárnév DEFINITIONS

5.4. A felhasználói szótár használata

Hatására az új definiciók ezután a "szótárnév" nevű felhasználói szótárba fognak kerülni, vagyis a "szótárnév" szótár cime a CURRENT rendszerváltozóba töltődik.

A gyakorlatban a DEFINITIONS kiadását közvetlenül a számrendszer beállitása követi.

Tekintsünk egy példát az imént tanultak szemléltetésére! Ha visszaemlékszünk a 3.2.2. pontra, ott egy komplex aritmetika létrehozásával foglalkoztunk. Ott az újradefiniálás hatását vizsgáltuk. Ha ismét az eredeti aritmetikát akartuk használni, úgy meg kellett szabadulnunk a komplex műveletektől. Most megmutatjuk, miként lehet önálló felhasználói szótárba szervezni új műveleteinket. Először létrehozzuk a KOMPLEX nevű felhasználói szótárat:

```
VOCABULARY KOMPLEX IMMEDIATE ok
```

Ezután jelöljük ki ezt a szótárat a definiciók befogadására:

```
KOMPLEX DEFINITIONS DECIMAL ok
```

Irjuk be a + - stb. definiciót:

```
: + ... ; ok
: - ... ; ok
      .
      .
      .
```

Most szavaink bekerültek a KOMPLEX szótárba, és használhatók is. Ha közben áttértünk más szótárra, akkor az ismételt használathoz aktivizálnunk kell az új szótárat:

```
KOMPLEX ok
```

Ha ismét az alapszótár + - stb. utasitását akarjuk

használni, egyszerűen vissza kell kapcsolni az alapszótárra, amelyet a következő pontban részletezünk.

5.4.3. Előre létrehozott szótárak a FORTH-ban

A FORTH rendszerben létezik három, előre létrehozott szótár, ezek a következők: a FORTH alapszótár, az assemblert tartalmazó szótár és az editor szótár. Felhasználásuk módja teljesen olyan, mintha magunk definiáltuk volna a VOCABULARY utasitás segitségével.

5.4.3.1. A FORTH alapszótár

Ez a rendszer inditásakor keresésre és definiciók befogadására kijelölt szótár. Az összes többi, beépitett vagy általunk létrehozott szótár ehhez kapcsolódik, a keresés ebben fejeződik be. Más szótárból az alapszótárba a **FORTH** szóval kapcsolhatunk vissza.

5.4.3.2. A beépitett assembler

Könyvünk során több helyen emlitettük, hogy gépi kódú törzzsel rendelkező szavak is létrehozhatók a beépitett assembler segitségével, de az eljárást konkrétan nem ismertettük. Természetesen maga az assembler minden processzoron más és más, használatát minden esetben az adott gépen futó FORTH felhasználói leirásából kell megtanulni. Ezért itt csak a közös vonásokat: az assembly betétet tartalmazó szó létrehozásának módját és az assemblerekben megtalálható magas szintű eszközöket mutatjuk be. A példákban mi az I8080 assembler mnemonokjait használjuk, más processzorokon nyilván ezektől eltérő jelölést kell alkalmazni. A fordító használata előtt az azt tartalmazó screeneket be kell olvasni, és a szótárt az **ASSEMBLER** szóval keresésre ki kell jelölni.

5.4. A felhasználói szótár használata

A gépi kódú törzzsel rendelkező szót két módon hozhatjuk létre. Az első mód: a szódefinicióban a kettőspont helyett a **CODE**, a pontosvessző helyett a **C;** szót használjuk, a szó törzse pedig assembly és FORTH utasitásokat tartalmaz.

CODE NÉV utasitások C;

A másik módszer a **;CODE** szó használata:

: NÉV ... ;CODE utasitások C;

Itt a ;CODE hatására befejeződik a NÉV definiciója, és sor kerül a ;CODE és C; közötti utasitások gépi kóddá forditására. A NÉV és a CODE között valamilyen "kreáló" utasitássorozatnak kell lennie (például <BUILDS-nek). Ezen utasitássorozat a <BUILDS és DOES> közötti utasitásokhoz hasonló feladatokat végezhet. A DOES> és a CODE között mindössze annyi a különbség, hogy az előbbinél FORTH-ban, az utóbbinál ASSEMBLER formában irhatjuk meg a végrehajtási algoritmust. Ezután a

NÉV GÉPIKOD

végrehajtása a következőt eredményezi: előbb lefut NÉV ;CODE-ig tartó (FORTH-ban irt) törzse, majd létrejön a GÉPIKOD nevű szó, amelynek végrehajtása az adott gépi kód végrehajtását jelenti. Ezt a módszert ritkábban alkalmazzuk, mint a CODE ... C; szerkezetet.

Lássuk most, mik lehetnek az " utasitások ". Először is lehetnek tetszőleges FORTH szavak. Ezek (függetlenül attól, hogy közvetlenek-e vagy sem) azonnal végrehajtódnak. Általában cimek kiszámitását végzik. Lehetnek assembly utasitások, amelyeket a FORTH filozófiájának megfelelően a forditott lengyel formátumhoz hasonlóan kell leirni. Erre néhány példát is mutatunk (I8080 utasitások).

szokásos irásmód	FORTH irásmód
POP H	H POP
MOV C,A	A C MOV
MVI A,7FH	7F A MVI
JMP 3000H	3000 JMP

Az összes többi utasitás hasonló módon képezhető. Emellett lehetőség van a következő magas szintű szerkezetek használatára:

IF..ELSE..THEN
BEGIN..WHILE..REPEAT
BEGIN..AGAIN

Ezek nem tévesztendők össze a FORTH hasonló szerkezeteivel. Ezekből gépi kódú utasitások fordulnak: feltételes vagy feltétel nélküli ugrások. A vezérlő feltételek az adott processzor lehetséges feltételvizsgálatai. Az i8080 esetén például CS (if Carry Set), PE (if Parity Even), 0=, 0<. Ezek ellentetjeit a NOT assembler operátorral állithatjuk elő. A keletkező vezérlési szerkezet viszont a FORTH-belivel egyezik meg.

A gépi kódú szó végén minden esetben egy ugrásnak kell lennie arra a cimre, amely a FORTH NEXT nevű rendszerkonstansában található. Ez az ugrás teszi lehetővé a FORTH program folytathatóságát.

Tekintsünk végezetül egy példát I8080 assemblyben megirva! Készitsünk egy KIIR nevű szót, amely a vermen levő cimtől kezdődően 0 bájtig kiirja a képernyőre az ott tárolt szöveget! Ehhez felhasználjuk a CP/M karakterkiiró szubrutinját (a használt assembler egy CP/M operációs rendszer alatt futó I8080 FORTH része). Ez a következőképpen hivható meg: a karaktert az E regiszterbe kell tölteni, a C

regiszterbe 6-ot kell irni, ez jelzi, hogy karakterkivitelt szeretnénk végrehajtani. Ezután a CP/M 5-ös cimen levő rutinját kell meghivni. Előbb lássuk ezt a programrészletet I8080 assemblyben megirva.

```
ENTRY EQU   5      ; CP/M belépési pont
KIIR: POP   H      ; kezdőcim a HL reg. párba
CIKL: MOV   A,M    ; karakter az A-ba
      ANA   A      ; flagek beállitása
      JZ    VEGE   ; ha A nulla volt, rutin vége
      MOV   E,A    ; E-be a karakter
      MVI   C,6    ; kivitel jelzése
      PUSH  H      ; HL mentése, mert az BDOS használja
      CALL  ENTRY  ; BDOS rutin hivása
      POP   H      ; HL visszaállitása
      INX   H      ; cim növelése
      JMP   CIKL   ; következő karakter kivitele
VEGE: JMP   NEXT   ; rutin vége, a FORTH a következő
                   ; szóval folytatja a futást
```

Ugyanez a beépitett assemblerrel igy alakul:

```
5 CONSTANT ENTRY      ( CP/M belépési pont )
CODE KIIR ( cim --- )
           H POP      ( kezdőcim a HL reg. párba )
    BEGIN
         M A MOV      ( karakter az A-ba )
           A ANA      ( flagek beállitása )
         0= NOT       ( feltételképzés )
    WHILE
      A E MOV         ( E-be a karakter )
      6 C MVI         ( kivitel jelzése )
        H PUSH        ( HL mentése )
    ENTRY CALL        ( BDOS-rutin hivás )
        H POP         ( HL visszaállitása )
        H INX         ( cim növelése )
```

```
        REPEAT
            NEXT JMP      ( a FORTH program futása a )
                          ( következő szóval folytatódik )
    C;
```

Most pedig bemutatjuk egy i8080 assembler screenjeit:

```
SCR # 1
 0 ( 8080 assembler, John J. Cassady )
 1 VOCABULARY ASSEMBLER IMMEDIATE ASSEMBLER DEFINITIONS
 2 HEX ' ASSEMBLER CFA ' ;CODE 08 + !
 3 NEXT-LINK CONSTANT NEXT
 4 : CODE ?EXEC CREATE !CSP ; IMMEDIATE
 5 : C; ?EXEC ?CSP SMUDGE ; IMMEDIATE
 6 : LABEL ?EXEC 0 VARIABLE SMUDGE -2 ALLOT [COMPILE]
 7   ASSEMBLER !CSP ; IMMEDIATE
 8 : 8* DUP + DUP + DUP + ;
 9 4 CONSTANT H 5 CONSTANT L 7 CONSTANT A 6 CONSTANT PSW
10 2 CONSTANT D 3 CONSTANT E 0 CONSTANT B 1 CONSTANT C
11 6 CONSTANT M 6 CONSTANT SP
12 : 1MI <BUILDS C, DOES> C@ C, ;
13 : 2MI <BUILDS C, DOES> C@ + C, ;
14 : 3MI <BUILDS C, DOES> C@ SWAP 8* + C, ;
15 : 4MI <BUILDS C, DOES> C@ C, C, ; -->
```

```
SCR # 2
  0 ( 8080 assembler, folytatas )
  1 : 5MI <BUILDS C, DOES> C@ C, , ;
  2 00 1MI NOP  76 1MI HLT  F3 1MI DI   FB 1MI EI
  3 07 1MI RLC  0F 1MI RRC  17 1MI RAL  1F 1MI RAR
  4 E9 1MI PCHL F9 1MI SPHL E3 1MI XTHL EB 1MI XCHG
  5 27 1MI DAA  2F 1MI CMA  37 1MI STC  3F 1MI CMC
  6 80 2MI ADD  88 2MI ADC  90 2MI SUB  98 2MI SBB
  7 A0 2MI ANA  A8 2MI XRA  B0 2MI ORA  B8 2MI CMP
  8 09 3MI DAD  C1 3MI POP  C5 3MI PUSH 02 3MI STAX
  9 0A 3MI LDAX 04 3MI INR  05 3MI DCR  03 3MI INX
 10 0B 3MI DCX  C7 3MI RST  D3 4MI OUT  DB 4MI IN
 11 C6 4MI ADI  CE 4MI ACI  D6 4MI SUI  DE 4MI SBI
 12 E6 4MI ANI  EE 4MI XRI  F6 4MI ORI  FE 4MI CPI
 13 22 5MI SHLD 2A 5MI LHLD 32 5MI STA  3A 5MI LDA
 14 C4 5MI CNZ  CC 5MI CZ   D4 5MI CNC  DC 5MI CC
 15 E4 5MI CPO  EC 5MI CPE  F4 5MI CP   FC 5MI CM -->
```

```
SCR # 3
  0 ( 8080 assembler, folytatas )
  1 CD 5MI CALL
  2 C0 1MI RNZ  C8 1MI RZ   D0 1MI RNC  D8 1MI RC
  3 E0 1MI RPO  E8 1MI RPE  F0 1MI RP   F8 1MI RM
  4 C9 1MI RET  C3 5MI JMP
  5 C2 CONSTANT 0= D2 CONSTANT CS E2 CONSTANT PE
  6 F2 CONSTANT 0< : NOT 8 + ; : MOV 8* 40 + + C, ;
  7 : MVI 8* 6 + C, C, ; : LXI 8* 1+ C, , :
  8 : ENDIF 2 ?PAIRS HERE SWAP ! ; : THEN [COMPILE] ENDIF ;
  9 : IF C, HERE 0 , 2 ;
 10 : ELSE 2 ?PAIRS C3 IF ROT SWAP ENDIF 2 ;
 11 : BEGIN HERE 1 ; : UNTIL SWAP 1 ?PAIRS C, , ;
 12 : AGAIN 1 ?PAIRS C3 C, , ; : WHILE IF 2+ ;
 13 : REPEAT >R >R AGAIN R> R> 2 - ENDIF ;
 14
 15 FORTH DEFINITIONS DECIMAL ;S
```

5.4.3.3. A beépitett szövegszerkesztő

A szövegszerkesztő screenjeit betöltjük a memóriába, majd az **EDITOR** szó kiadásával aktivizáljuk a szótárat. Az editorral általában az aktuális screent tudjuk módositani. Ezt azt jelenti, hogy először valamilyen módon (célszerűen a LIST parancs kiadásával) be kell hoznunk a memóriába a kivánt sorszámú screent. Ezután elkezdhetjük az aktuális screen editálását. Az editorfunkciók igen eltérőek lehetnek, igy ezeket a konkrét FORTH megvalósitások leirásánál, az F3. függelékben ismertetjük.

6. A FORTH rendszer működése

Az előző öt fejezetben a FORTH szinte teljes eszköztárát megismertük már. A leirt szavak segitségével a gyakorlatban felmerülő feladatok több, mint 90 százaléka biztosan megoldható. A nyelv rendelkezésünkre bocsátja mindazokat a szolgáltatásokat, amelyek általában megtalálhatók egy magas szintű programozási nyelvben.

Ebben a fejezetben a FORTH rendszer gyökeréig fogunk hatolni, a FORTH legrejtettebb működésével ismerkedünk meg. A fejezet megértéséhez feltétlenül szükséges bizonyos fokú jártasság a gépi kódú programozásban. A fejezet során mindenesetre nem egy létező processzor utasitásait fogjuk használni, hanem egy egyszerűsitett jelölési módszert alkalmazunk. Ez a következő:

a) 16 bites cimeket és regiszteret feltételezünk. A regiszterek különféle cimek tárolására szolgálnak, és a későbbiekben ezeknek majd nevet is adunk, amellyel hivatkozni lehet rájuk.

b) Magában álló regiszternév, például R a regiszterben levő 16 bites cimet reprezentál. Ha a regiszternév valamilyen művelet operandusa, akkor a műveletet a regiszter tartalmával kell elvégezni. Tehát, ha R-ben 123 van, akkor R+2-t 125-ként kell értelmezni.

c) Zárójelbe tett regiszternév az indirekt cimzés jele. Ha R-ben 123 van, akkor (R) a 123-as cimen található alsó és a 124-es cimen található felső bájtokból alkotott 16 bites értéket jelent.

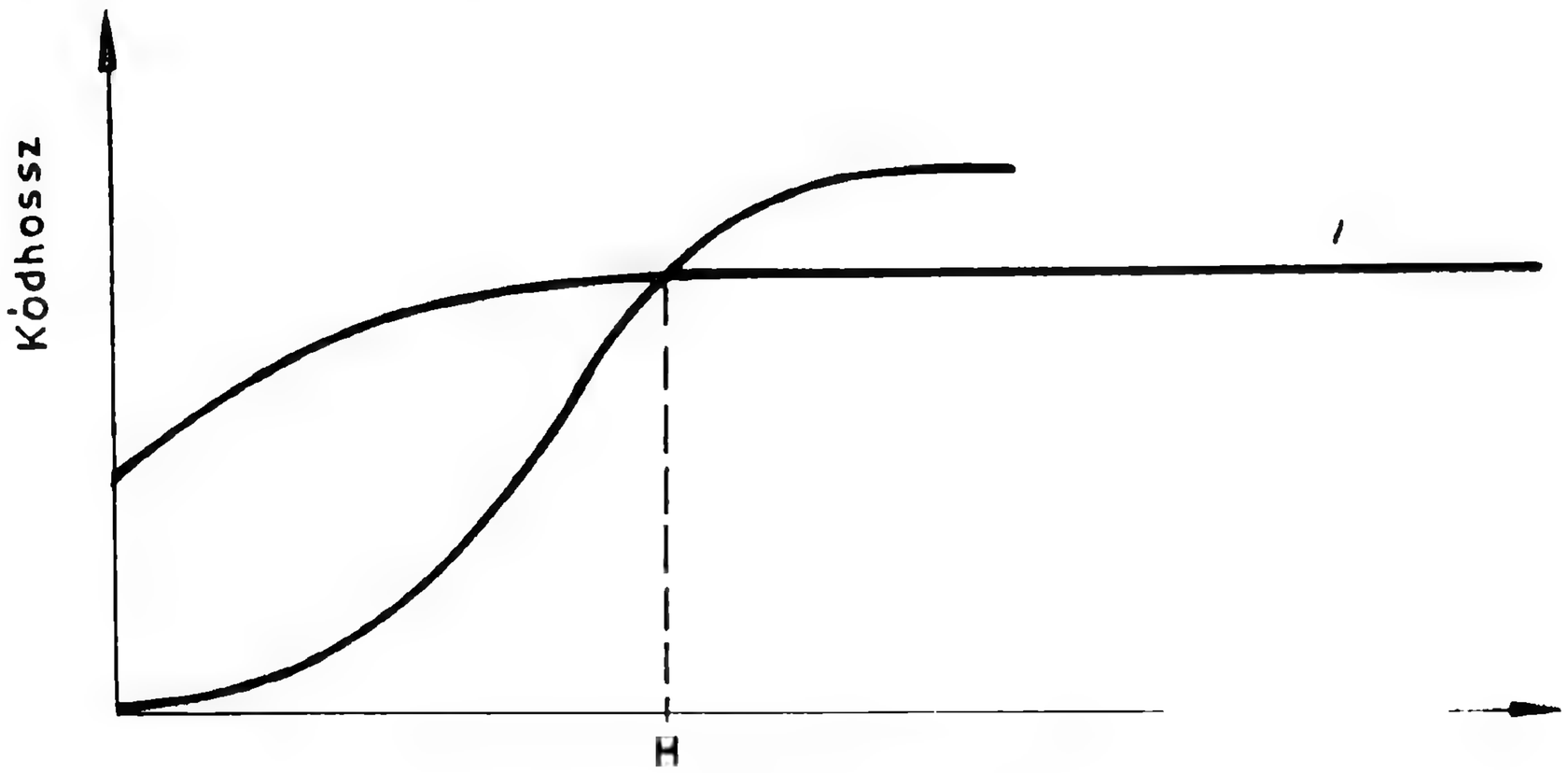
d) A <-- jellel az adatmozgatást jelöljük. A nyil jobb oldalán levő kifejezés értéke a bal oldalon álló

regiszterbe vagy regiszter által indirekten kijelölt memóriaszóba iródik. Tehát R1 <-- (R0) azt jelenti, hogy az R0-ban levő cimen található 16 bites adat az R1 regiszterbe kerül.

e) A CALL a szubrutinhivást, a JUMP az ugrást jelöli. A cimkéket a gépi kódú programozásban megszokott értelemben használjuk.

6.1. A FORTH kódhossza és futási ideje

Már az előző részekben is emlitettük, hogy a FORTH igen tömör kódot állit elő. Azt is tudjuk, hogy ez nem valami leforditott, gépi kódú program, hanem maga a szótár. A szavak végrehajtása a szótárban tárolt cimek alapján történik. Hasonlitsuk össze most a FORTH kód (azaz a szótárhossz) és egy, ugyanazt a feladatot assembly nyelven megoldó program hosszát (6.1. ábra).



6.1. ábra
A kódhossz az intelligencia függvényében

A program intelligenciáján az elvégzendő alapfunkciók számát értjük. Az alapfunkció nem feltétlenül egy szó vagy

assembly nyelv esetén egy szubrutin, hanem lehet ezek logikailag összetartozó, egy konkrét részfeladat megoldására szolgáló csoportja. (A fenti ábra inkább minőségi, mint mennyiségi összefüggést mutat.)

Láthatjuk, hogy gépi kód esetén egy bizonyos határig további alapfunkciók beépitése jelentős kódnövekedéssel jár. Ezen határ elérése után általában az exponenciális jellegű növekedés logaritmikussá megy át. Ennek magyarázata az, hogy egyre több, már megirt alprogramot vagy alapfunkciót tudunk felhasználni, és a program továbbfejlesztése ezen a szinten főleg adatelőkészitésből és szubrutinhivásokból áll.

A FORTH-nál más a helyzet. Itt a nulla intelligenciához is egy adott kódhossz tartozik: az alapszótár. Ezután rövid, viszonylag meredek emelkedés után beáll egy meglehetősen lapos szakasz. Jól látható, hogy van egy H határ, amely alatt a gépi kódú program a rövidebb. Igy, ha egy feladat megoldásánál a kódhossz a döntő tényező, akkor bizonyos esetekben az assembly program az előnyösebb. Ha viszont a programfejlesztés kényelmét tartjuk szem előtt, akkor a FORTH messze megelőzi az assemblereket.

Ugyanezt az összehasonlitást megtehetjük a magas szintű nyelvekkel is (PASCAL, FORTRAN, C stb.). Ezen nyelvek forditóprogramjai az esetek nagy részében gépi kódú programokat állitanak elő. Az igy keletkező programok kódhossz szempontjából igen jónak mondhatók (leegyszerűsitve a dolgot: nem sokkal hosszabbak annál, ahogy egy átlagos programozó assembly nyelven megirmá őket). A kód tovább rövidithető a mindig meglevő redundanciák kiküszöbölésével. Erre szolgálnak a forditókhoz tartozó optimalizálók, amelyek a felesleges kódrészek kiszűrését végzik. Eredményképpen tehát a magas szintű nyelvekből forditott gépi kódú programok hossza körülbelül a 6.1. ábra assembly-vonalának megfelelően alakul. Ezért ezekre is van

egy határ, amely felett a FORTH sokkal rövidebb kóddal oldja meg az adott feladatot.

Más a helyzet a futási idő terén. Itt a FORTH már nem dicsekedhet olyan kimagasló eredményekkel, mint a kódhossz terén. A gépi kódú programok természetes gépi sebességgel futnak. A FORTH-nak viszont egy speciális (a későbbiekben ismertetendő) futtatási algoritmusa van. Ezen algoritmus futási ideje minden szó végrehajtása esetén (mint azt látni fogjuk, akár többször is) hozzáadódik a szótörzs végrehajtási idejéhez. Emiatt, főleg sok szóból álló program esetén már jelentős sebességcsökkenéssel számolhatunk. Sok esetben persze ez sem jelent hátrányt, hiszen a legtöbb FORTH-ban megtalálható beépitett assembler segitségével gépi kódú szavakat is gyárthatunk. Ezek elláthatják a sebesség-kritikus feladatrészek megoldását.

Hogy a FORTH és más nyelvek futási idejét összehasonlithassuk, egy igen egyszerű utasitássorozat futási idejét mértük meg két különböző FORTH rendszer, egy BASIC interpreter és egy PASCAL fordító felhasználásával.

A hardver környezet egy Z80A mikroprocesszorra épülő fejlesztőrendszer volt. A mikrogépen az igen elterjedt CP/M operációs rendszer felügyelete alatt futottak a programok. A processzor 2.5 MHz-es órajellel működött.

A programok funkciója mind a négy esetben azonos volt: egy nullától 32000-ig futó egyszerű ciklus. A ciklusmag a ciklusváltozó modulo 2-es osztását végezte. Először a "Microsoft Z80 FORTH 2.04" FORTH-on futtattuk le a CIKLUS nevű szót, amelynek definiciója itt látható.

6.1. A FORTH kódhossza és futási ideje

```
: CIKLUS 32000 0 DO I 2 MOD DROP LOOP ;
```

A szó futása 1 perc 14 másodpercet vett igénybe.

A következő a szerzők által készitett, valós idejű feladatokat is megoldó műszervezérlő FORTH volt (a függelékben részletesen ismertetni fogjuk). A rendszer neve "SIF-Z80 FORTH 2.2". A futtatott szó itt is a CIKLUS volt, definiciója megegyezik a fentebb közölttel.

A szó futási ideje: 1 perc 6 másodperc.

Ezután a "Microsoft BASIC-80 Rev. 5.2." Basic interpreteren hajtattuk végre az alábbi sort:

```
10 FOR I=1 TO 32000 : J=I MOD 2 : NEXT I
```

A végrehajtási idő 3 perc 9 másodpercre adódott.

Végül a MicroSYSTEMS, Inc. "PASCAL/MT+ RELEASE 5.5." Pascal forditóját használtuk. A program a következő volt:

```
PROGRAM teszt(input,output);
VAR i,j:INTEGER;
BEGIN
  FOR i:=0 TO 32000 DO
    j:=i MOD 2
END.
```

A futás itt mindössze 31 másodpercet vett igénybe. A kapott időeredményeket összefoglalva tehát ezt kapjuk:

ó.1. A FORTH kódhossza és futási ideje

Z80 FORTH 2.04	74 sec.
SIF-Z80 FORTH 2.2.	66 sec.
BASIC-80	191 sec.
PASCAL/MT+	31 sec.

Láthatjuk tehát, hogy a FORTH majdnem háromszor olyan gyors, mint egy BASIC interpreter, annak ellenére, hogy a FORTH nem gépi kódra fordit. A PASCAL-nál viszont jóval lassabban dolgozik, a ciklus lefuttatása több, mint kétszer annyi időt vesz igénybe. érdekes, hogy még két FORTH között is van különbség, igaz mindössze 8 másodperc. Ez valószinüleg az utasitáselővételi algoritmus eltérő kódolásából adódik.

6.2. A "threaded code" fogalma

A következőkben megismerkedünk azzal a programozás-technikai módszerrel, amelyet a FORTH használ a kódtömörités érdekében. Tegyük fel, hogy rendelkezésünkre áll egy szubrutincsomag. A szubrutinok bemenő paramétereiket a vermen vagy kijelölt memóriarekeszekben várják, kimenő paramétereiket pedig ugyanitt helyezik el. Ebben az esetben egy adott feladat megoldásához sorban meg kell hivni valahány szubrutint. A program tehát igy fog kinézni (ó.2. ábra):

6.2. A "threaded code" fogalma

```
START:  CALL   SZUB1
        CALL   SZUB2
                 .
                 .
                 .
        CALL   SZUBn
        CALL   VEGE
```

6.2. ábra
Program felépitése szubrutinokból

A SZUB1, SZUB2, ... SZUBn alprogramok a feladat megoldását végzik. A VEGE nevü szubrutin leállitja a program futását (például visszatér az operációs rendszer monitor-programjába).

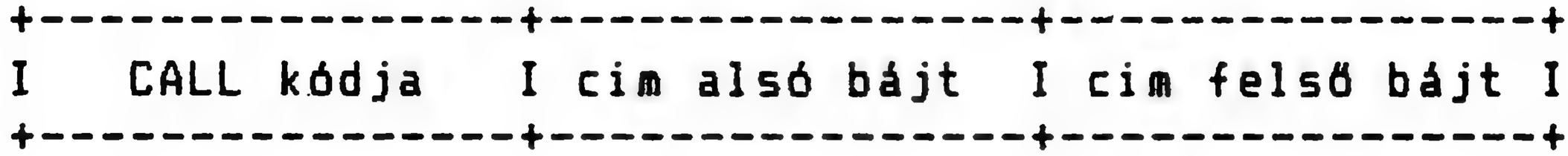
Természetesen a programfejlesztés elképzelhető úgy is, hogy előbb kisebb alprogramokat irunk az eredeti csomag felhasználásával (azaz egy új, saját igényeinknek megfelelő csomagot hozunk létre), majd ennek segitségével állitjuk össze a főprogramot. A program tehát ilyen alakot is ölthet (6.3. ábra).

```
START:  CALL   SZUB1
        CALL   SZUB2 -----> SZUB2:  CALL   SZUB20
                                    CALL   SZUB21
             .                        .
             .                        .
             .                        .
        CALL   SZUBn                  .
        CALL   VEGE                 CALL   VISSZA
```

6.3. ábra
Program felépitése szubrutinokból,
saját csomag felhasználásával

A VISSZA nevű alprogram a visszatérést (RETURN utasitást) valósitja meg. Mivel megállapodás szerint programunkban csak szubrutinhivást használhatunk, nem irhatjuk le egyszerűen a RETURN-t.

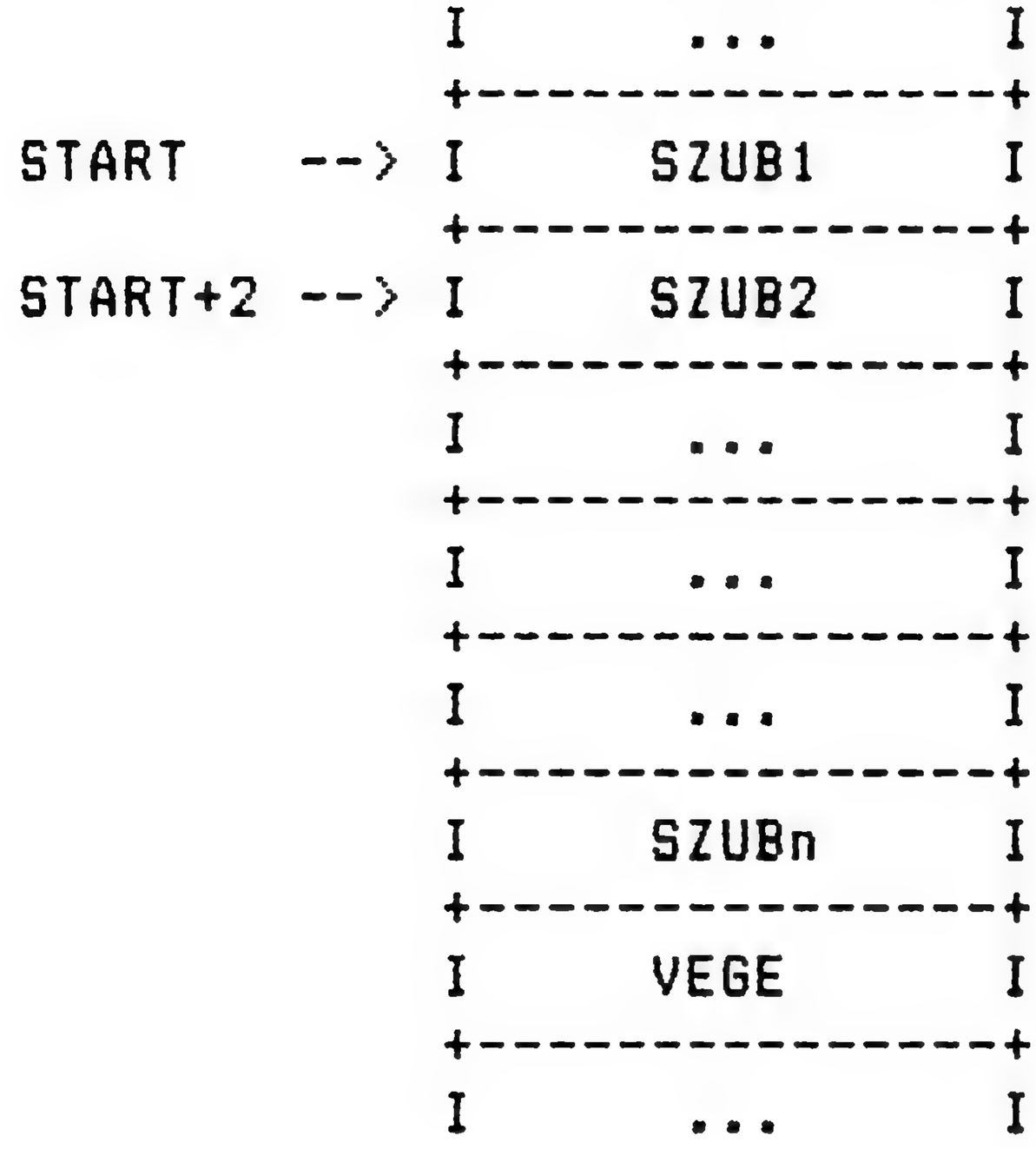
Lássuk, milyen lehetőség kinálkozik a keletkező kód hosszának csökkentésére! A 6.2. vagy 6.3. ábrán látható programrészletek egy-egy sorából a legtöbb processzoron az alábbi kód áll elő (6.4. ábra):



6.4. ábra
Egy szubrutinhivásból keletkező kód

Ez tehát három bájt. Az egyes sorokból keletkező kódban mindig megtalálható a CALL utasitás kódja. Miért ne hagynánk el ezt a felesleges bájtot? Tényleges információt nem hordoz,

hiszen tudjuk, hogy minden egyes cim előtt ott 'kell lennie. Elhagyásával, főleg hosszú program esetén jelentős megtakaritásra tehetnénk szert: hivásonként egy bájtra. Tekintsük most először a 6.2. ábra programját! Hagyjuk el a CALL utasitásokat, és tároljuk a szubrutinok cimeit egy táblázatban (6.5. ábra).



6.5. ábra
A szubrutincimekből álló táblázat

Az igy keletkező szerkezetet, amely tehát egymás után végrehajtandó szubrutinok hivási cimeit tartalmazza, **threaded code-nak, láncba főzött vagy felfőzött kódnak** nevezzük. A fenti struktúra kisértetiesen hasonlit a FORTH szótár szótörzseire (a meghatározásrészre), amely szintén szavak

cimeinek listája. Valóban, a FORTH szótár is egy felfűzött kód. A threaded code-ot egyébként nem csak a FORTH esetében használják. Vannak például olyan FORTRAN forditók is, amelyek szintén felfűzött kódot állitanak elő.

6.3. A FORTH rendszer mint virtuális processzor

Ismeretes, hogy egy valódi processzor hogyan hajtja végre a memóriában tárolt programot. Először kiolvassa a memóriából a következő bájtot vagy szót, amely a végrehajtandó utasitás kódját és esetleges paramétereit tartalmazza. Ezt nevezzük utasitás-elővételi ciklusnak. Az utasitáskódot a processzor dekódolja, és eldönti, szükség van-e további memóriaolvasásra (vagy olvasásokra). Ha igen, elvégzi ezeket is. Ilyenkor a processzor az utasitás további paramétereit, adatokat vagy cimeket olvas be. Az összes szükséges adat birtokában elvégzi az utasitáskód által kijelölt műveleteket, s az eredményt, ha van, valamelyik regiszterébe vagy a memóriába irja. Ezután rátér a következő utasitás beolvasására, azaz újabb utasitás-elővételi ciklust generál.

A FORTH is tulajdonképpen hasonlóan dolgozza fel a szótárt alkotó felfűzött kódot: felismeri a memóriában soron következő szócimet (hogy hogyan, azt a következő pont mutatja be). Ez megfelel az utasitás-elővételi ciklusnak. Magát a szócimet akár utasitáskódnak is felfoghatjuk. Ha például a MIN szó az A97E hexadecimális cimen kezdődik, akkor mondhatjuk, hogy a FORTH számára a MIN kódja A97E. Ezek alapján nyugodtan állithatjuk, hogy a felfűzött kódon dolgozó FORTH rendszer olyan, mint ha egy processzor lenne: a FORTH egy **virtuális processzor**. Ennek a virtuális processzornak a szócimek az utasitáskódjai, utasitásai pedig a szavakhoz rendelt működések. A FORTH processzor utasitáskészlete új

szavak definiálásával tetszőlegesen bővithető. Ismerkedjünk meg most részletesen a FORTH utasitás-elővételi algoritmusával!

6.4. Az utasitás-elővételi algoritmus

Első lépésként konstruáljuk meg azt a programrészletet, amely a 6.5. ábrán látható felfűzött kódot alkotó szubrutinok egymás utáni végrehajtását eredményezi! Tegyük fel, hogy rendelkezünk egy MUTAT nevű regiszterrel. A MUTAT-ban azt a cimet fogjuk tárolni, amely cimen az éppen végrehajtandó szubrutin cime van. A programrészlet a 6.6. ábrán látható, a fejezet elején bevezetett egyszerű jelöléseket felhasználva.

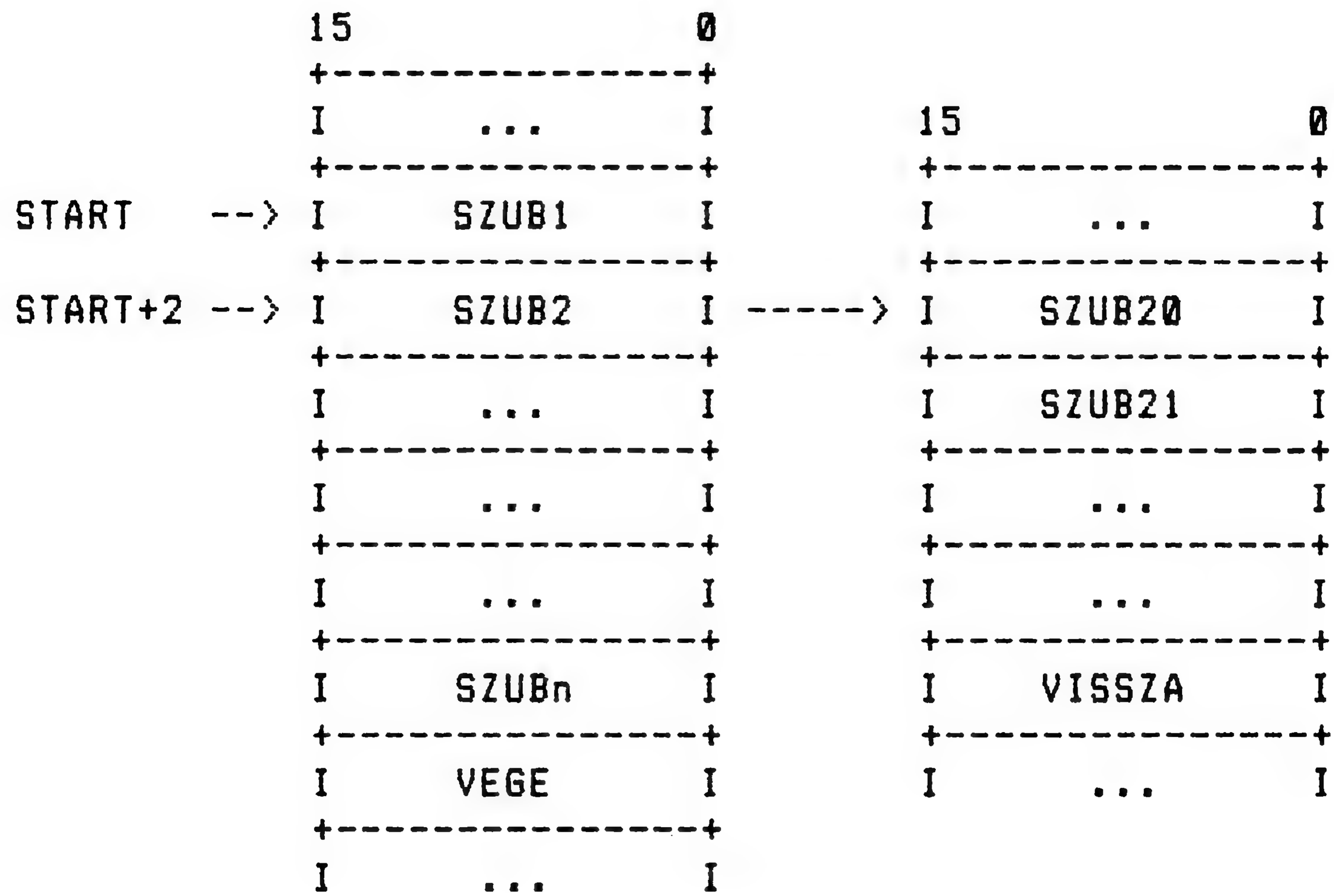
```
          MUTAT <-- START-2
KOVETK:   MUTAT <-- MUTAT+2
          CALL (MUTAT)
          JUMP KOVETK
```

6.6. ábra
Programrészlet a felfűzött kód szubrutinjainak végrehajtására

Az algoritmus működése a következő: az első sorban beállitjuk a MUTAT kezdőértékét. A MUTAT igy az első végrehajtandó szubrutincimet tartalmazó memóriaszó elé mutat. A KOVETK cimkén található utasitás a regiszter tartalmát kettővel megnöveli. Emiatt a MUTAT-ban most az a cim van, amely cimen a SZUB1 cimét helyeztük el (tehát a START nevű memóriaszó). A harmadik sor egy regiszter indirekt szubrutinhivás. Ennek hatására SZUB1 végrehajtódik. A

szubrutinból való visszatérés után a vezérlés a negyedik sorra adódik, amely egy ugrás a KOVETK cimkére. Igy újabb utasitás-elővételi ciklus generálódik, amely a láncbafűzött kódot alkotó következő szubrutin végrehajtását eredményezi. Ez az eljárás mindaddig folytatódik, amig a VEGE rutin elérése a folyamatot meg nem szakitja.

A 6.6. ábrán bemutatott algoritmus mindaddig működőképes, amig a felfűzött kód egyes szubrutinjai valóban gépi kódban irt alprogramok. Állitsuk elő azonban a 6.3. ábrán közölt programváltozatból kapható láncba fűzött kódot. Ez a 6.7. ábrán látható.



6.7. ábra
A szubrutincimekből álló táblázat. A SZUB2 szintén felfűzött kódra mutat

6.4. Az utasitás-elővételi algoritmus

Nyilvánvaló, hogy a 6.6. ábra programja ennek a feladatnak megoldására nem alkalmas. A SZUB2 esetén a CALL (MUTAT) utasitással a felfűzött kódból álló alprogramot mint gépi kódú rutint hajtanánk végre, ami igen meglepő eredménnyel járna.

Meg kell tehát oldanunk a láncba fűzött kódon belüli szubrutinhivást. Ehhez természetesen szükséges, hogy a MUTAT regiszterrel rá tudjunk mutatni a felfűzött kód mellékágára, majd vissza tudjunk állni a főág megfelelő pontjára. Ezt esetleg többszörös mélységben is meg kell tenni, hiszen például a SZUB21 is lehet lácbafűzött kód. és persze fel kell ismerni azt is, hogy mikor értük el a valóban végrehajtható gépi kódot.

Az előttünk álló problémát a következő hasonlattal tehetjük szemléletessé: tegyük fel, hogy egy ódon várban meg kell találnunk a királylányt (ez jelenti a feladat megoldását). A vár szobáit (memóriarekeszek) sorban vizsgáljuk végig. A szobában vagy sárkány van (gépi kód), vagy kulcsok a szobából nyiló többi szobához (felfűzött kód). Feladatunk kettős. Egyrészt meg kell tudnunk különböztetni a sárkányt a kulcstól, hiszen a sárkány nem nyitja az ajtót, a kulcsot viszont nem tudjuk megölni. Másrészt, ha a szobában kulcsokat találunk sárkány helyett, akkor el kell kezdeni a mellékszobák szisztematikus vizsgálatát. Emellett mindig észben kell tartani, hogy honnan jöttünk, merre kell majd vissza-, illetve továbbmenni stb. A mellékszobák szövevényében addig kell haladnunk a kulcsok segitségével, amig sárkányt nem találunk. Ezt megöljük (végrehajtjuk a gépi kódú alprogramot), majd a már bejárt szobákon keresztül visszatérünk, és a következő még fel nem deritett szobán keresztül haladunk tovább. Ezt a bolyongást folytatjuk egészen a királylány megtalálásáig.

Pontosan ezt a feladatot kell a FORTH-nak megoldania.

6.4. Az utasitás-elővételi algoritmus

Ehhez a 6.6. ábra algoritmusát kell egy kicsit megváltoztatni, és néhány új fogalmat kell bevezetni.

6.4.1. Az IP és W regiszterek szerepe. A NEXT ciklus

A következőkben leirjuk a FORTH által használt utasitás-elővételi algoritmust. Ezután megmutatjuk, hogy az algoritmus miképpen működik akkor, ha a felfűzött kód következő kódeleme gépi kódú szóra mutat, és akkor, ha ugyanez egy újabb felfűzött kódot jelöl ki. Látni fogjuk azt is, hogy a bemutatott algoritmus helyes működéséhez a gépi, ill. felfűzött kódú szavak milyen felépitése szükséges.

A FORTH-ban a soron következő szó hivását szolgáló programrészlet neve **NEXT** (= következő) ami a szó végrehajtását két regiszter segitségével valósitja meg. Az egyik a FORTH virtuális processzor programszámlálója (PC=Program Counter), a neve IP (Instruction Pointer = utasitásmutató). A másikat W-vel jelölik és az algoritmus során mint segédregiszter szerepel. A NEXT-et a 6.8. ábrán láthatjuk.

```
NEXT:  IP <-- IP+2
       W  <-- (IP)
       JUMP (W)
```

6.8. ábra
A NEXT felépitése

Az algoritmus működőképességének feltételei gépi kódú szó esetén:

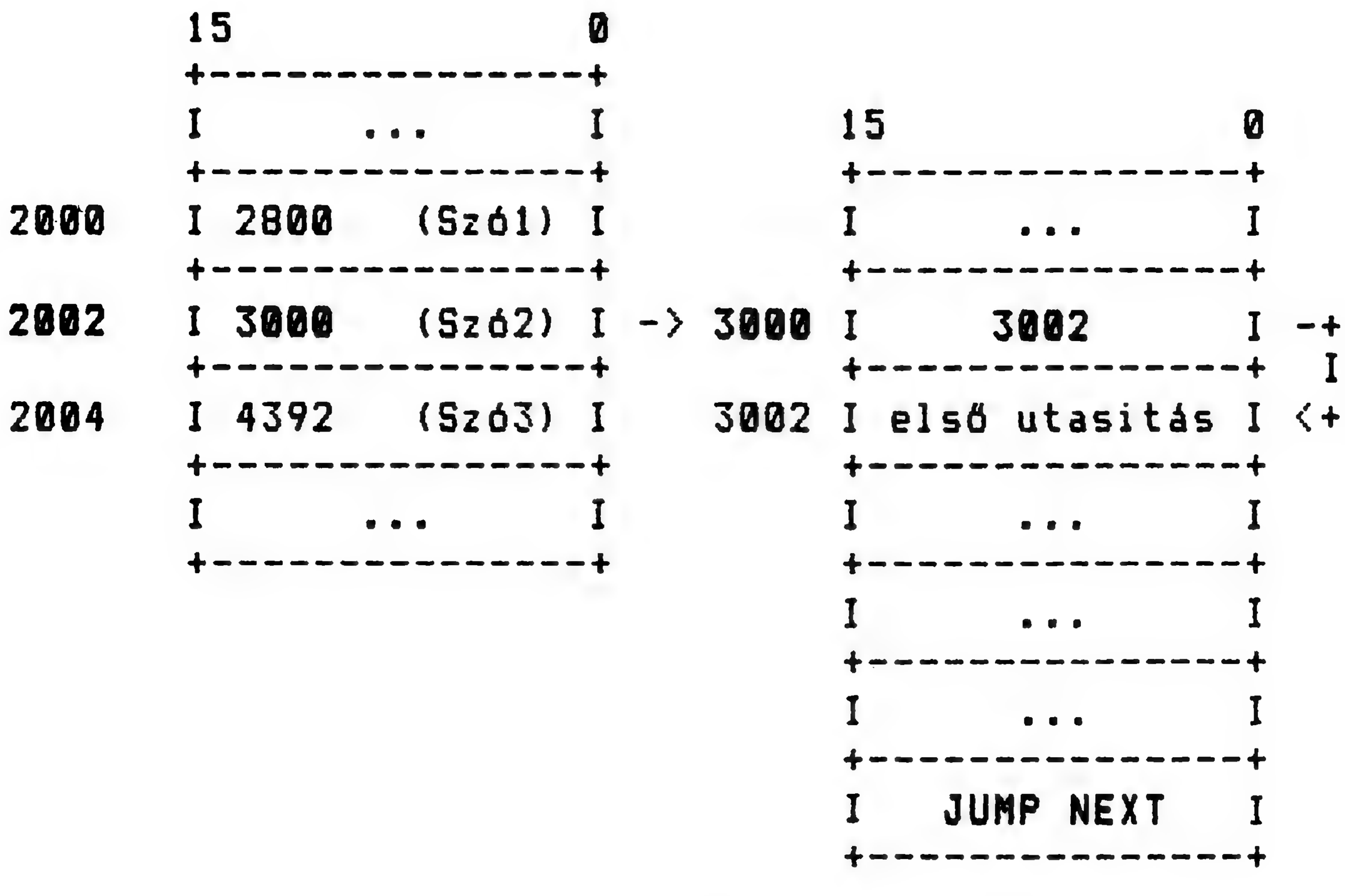
a) Abban a memóriaszóban, ahova a felfűzött kódban levő

szókezdőcim mutat, nem a gépi kódú szó első utasitásának kódja van, hanem az a cim, ahol ez a kód megtalálható. (A gyakorlatban ez általában a következő memóriaszó szokott lenni, de természetesen a szubrutin máshol is lehet a memóriában).

b) A szubrutin nem RETURN utasitással végződik. Az alprogram utolsó utasitása ehelyett JUMP NEXT.

Egy, a felsorolt követelményeknek is eleget tevő programrészletet látunk a 6.9. ábrán.

6.4. Az utasitás-elővételi algoritmus



ó.9. ábra
A NEXT működéséhez szükséges módon felépitett
gépi kódú szó

Tegyük fel, hogy a 2800-as cimen levő Szó1-et már végrehajtottuk és a program futása a NEXT-tel folytatódik. Az IP-ben 2000 van, hiszen ezen a cimen volt található a Szó1 cime, a 2800. A W tartalma érdektelen. A NEXT első utasitása hatására az IP tartalmához kettő hozzáadódik. Igy az IP-ben most 2002 található. A második sor egy IP indirekt töltés, a W-be tehát a 2002-es cimen található cim, azaz 3000 kerül. Az utolsó sor egy W indirekt ugrás, a vezérlés ily módon a W-ben levő cimen elhelyezkedő cimre adódik. A W-ben 3000 van, a 3000-es cimen 3002, ezért a program futása innen folytatódik. A 3002-es cimen viszont éppen a kivánt gépi kódú programrész (szótörzs) található, amely igy végrehajtódik. A törzs

lefutása után, annak utolsó, JUMP NEXT utasitásának hatására a vezérlés a NEXT-re adódik. Az IP megnövelődik kettővel, tartalma 2004 lesz, és igy biztositva van a következő, 4392-es cimen kezdődő Szó3 végrehajtásának kezdeményezése is.

Beláttuk tehát, hogy a ó.8. ábra algoritmusa képes a gépi kódban megirt szótörzseket végrehajtatni azok ismertetett szerkezeti felépitése esetén. Következő lépésként lássuk, mi módon érhetjük el, hogy a NEXT felfűzött kódú szótörzsek feldolgozását is lehetővé tegye.

6.4.2. A visszatérési verem szerepe. A DOCOL és a SEMIS

Azt már láttuk, hogy a FORTH virtuális processzorként működik. Most idézzük emlékezetünkbe azt, hogy miként kezeli egy valóságos processzor a szubrutinhivást! A szubrutinhivó CALL utasitás hatására a processzor a verembe menti a programszámláló aktuális értékét (és a legtöbb esetben egyéb más paramétereket is, de ennek most nincs jelentősége). A PC mentése biztositja, hogy a szubrutinból való visszatérés után rátaláljon a processzor arra a helyre, ahonnan a programot folytatni kell (tehát a hivás utáni utasitásra).

Ezt az eljárást a FORTH is átvette. A mentési verem szerepére az adatverem nem alkalmas, hiszen az kizárólag a szavak közötti adatforgalomra szolgál. Ezért van a FORTH-ban a visszatérési verem. Felfűzött kódú szótörzs feldolgozása előtt ide mentődik a FORTH programszámlálója, az IP regiszter. A mentés után az IP tehát felhasználható a szótörzsön mint mellékágon való végigléptetésre. Ily módon a NEXT felfűzött kódon való működéséhez a következő feltételek szükségeltetnek:

a) Abban a memóriaszóban, ahova a felfűzött kódban levő

kezdőcim mutat, nem a felfűzött kódú szó első kódeleme (azaz a szótörzset alkotó szavak közül az elsőnek a cime) van, hanem egy speciális, gépi kódban irt programrész, az úgynevezett **DOCOL** cime.

b) A DOCOL után levő felfűzött kódú szótörzs utolsó eleme a **;S** cime. A ;S-t a screenkezelésnél már megismertük. A ;S törzse gépi kódú, és ezt a kódrészletet **SEMIS**-nek szokták nevezni.

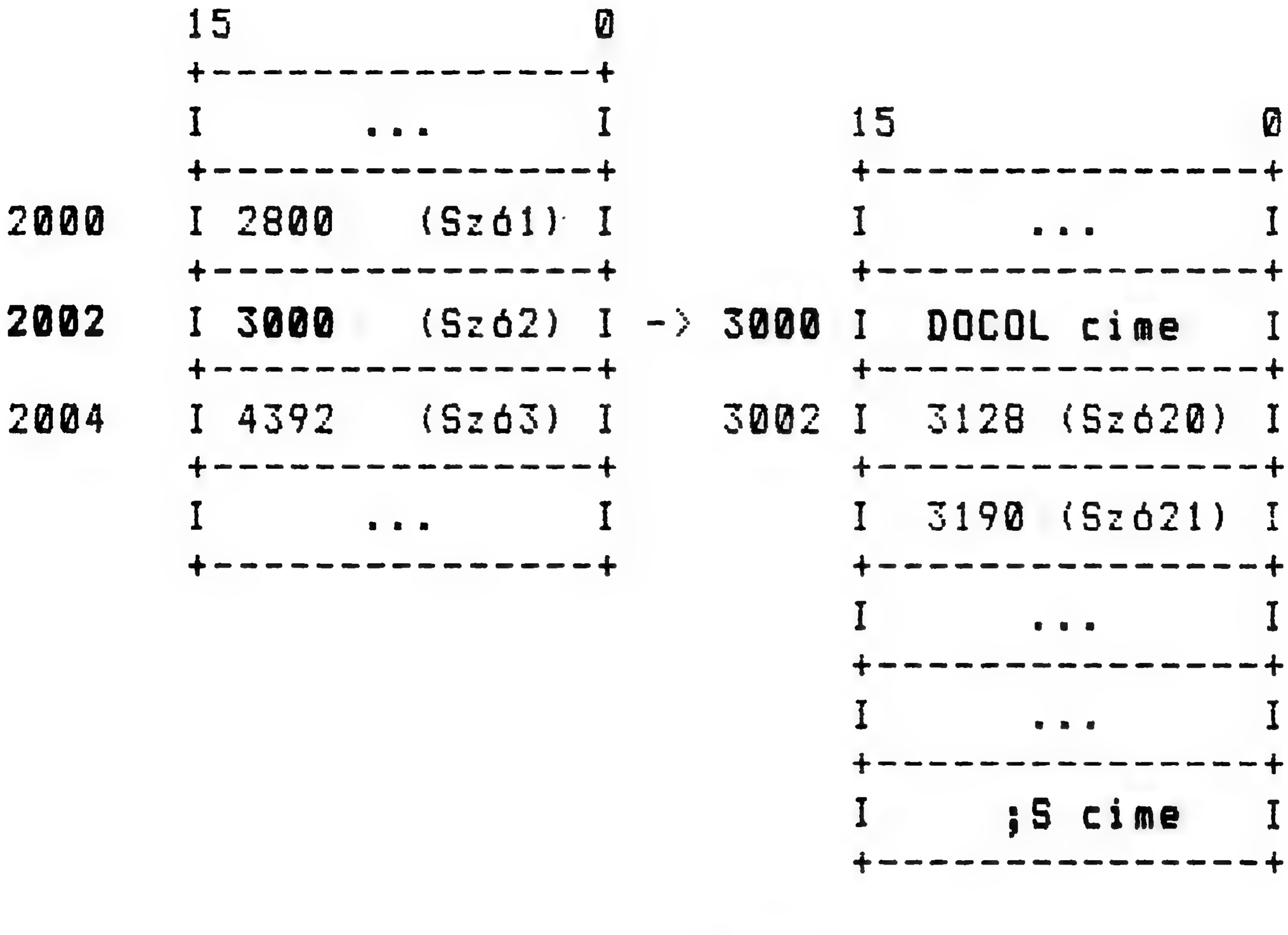
A 6.10. ábrán a DOCOL, a 6.11.-en a SEMIS felépitését láthatjuk. Az **RSP** regiszter a visszatérési verem mutatóját (Return Stack Pointer) jelöli. A 6.12. ábra egy helyesen felépitett, felfűzött kódú szóhivást mutat.

```
DOCOL:  RSP   <-- RSP-2
        (RSP) <-- IP
        IP    <-- W
        JUMP NEXT
```

6.10. ábra
A DOCOL felépitése

```
SEMIS:  IP  <-- (RSP)
        RSP <-- RSP+2
        JUMP NEXT
```

6.11. ábra
A SEMIS (a ;S törzse) felépitése



6.12. ábra
A NEXT működéséhez szükséges módon felépitett felfűzött kódú szó

Kövessük végig a működést erre az esetre. Mint az előző példában, most is tegyük fel, hogy a FORTH éppen a NEXT-et készül végrehajtani, az IP regiszterben 2000 van, a W tartalma most is érdektelen. Először az IP kettővel növelődik, igy tartalma 2002 lesz. Ezután a W-be töltődik a 2002-es cimen található cim, azaz a 3000. Végül egy ugrás történik a 3000 cimen levő cimre, azaz a DOCOL-ra. Most tehát az IP-ben 2002, a W-ben 3000 van, és megkezdjük a DOCOL végrehajtását. Először az RSP-ben levő cimből kettő levonódik, igy az most a visszatérési verem következő üres szintjére mutat. Ebbe a szintbe iródik be az IP aktuális

értéke, a 2002. Ezután az IP-be betöltődik a W-ben levő cim, azaz a 3000, majd a vezérlés a NEXT-re adódik. Itt az IP-hez kettő hozzáadódik, igy új értéke 3002 lesz. Tehát éppen a felfűzött kódú szótörzs első elemére mutat! Sikerült elérnünk azt, hogy mig az IP eredeti értéke, a 2002 a visszatérési vermen megörződik, addig az IP új tartalmával, a 3002-vel rá tudunk mutatni egy láncba fűzött kódú oldallánc első kódelemére. A ;S elérésekor a visszatérési veremből az IP-be iródik vissza a 2002, amely igy újból a főláncra mutat. Az RSP-hez kettő hozzáadódik, ezzel a 2002 tárolására szolgáló veremszint megszűnik. A program futása a NEXT-tel folytatódik.

Láttuk tehát, miképpen oldja meg a FORTH a szubrutin-kezdőcim és JUMP NEXT közé ágyazot gépi kódú, ill. a DOCOL és ;S közé ágyazott felfűzött kódú szótörzsek végrehajtását a NEXT segitségével. Könnyen belátható az is, hogy az ismertetett algoritmussal elméletileg végtelen, gyakorlatban a visszatérési verem véges mélysége miatt korlátozott számú hivási lánc feldolgozható. A hivási lánc elemei a 6.12. ábra szerintiek, azaz egy felfűzött kódú szótörzsnek legalább egy eleme újabb felfűzött kódú szó törzsére mutat. A lánc utolsó eleme természetesen mindig a 6.9. ábrán látható gépi kódú szó, hiszen a processzor csak ilyet tud végrehajtani.

Ne feledjük, hogy az emlitett keretek (DOCOL, ;S, JUMP NEXT stb.) szótörzsbeli elhelyezése a FORTH rendszer dolga, ezeket szódefinició közben a FORTH automatikusan beirja a szótárba.

Tekintsünk egy példasorozatot, amely az eddig elmondottakat próbálja illusztrálni! Legyen az alábbi három szó:

```
: 3. ." HARMADIK " ;
: 2. 3. ." MASODIK " ;
: 1. 2. ." ELSO " ;
```

Abban semmi meglepő nincs, hogy az 1. szó végrehajtása a következő eredményt adja:

```
1. HARMADIK MASODIK ELSO ok
```

Vegyünk be a 3.-ba két új szót. A három szó most így néz ki:

```
: 3. ." HARMADIK " R> DROP ;
: 2. 3. ." MASODIK " ;
: 1. 2. ." ELSO " ;
```

Próbáljuk az eddig megismertek alapján megjósolni, mit eredményez most az 1. szó végrehajtása! A FORTH végigfut a hivási láncon. A visszatérési veremre sorban a következő IP értékek kerülnek: előbb az 1. szó törzsében a 2.-ra mutató IP, majd a 2.-ban a 3.-ra mutató. A 3.-ban először kiiródik az, hogy HARMADIK. Ezután az R> szóval mintegy becsapjuk a FORTH-t, hiszen levesszük a visszatérési veremről a 2. szóba való visszatéréshez szükséges IP értéket. Ezért a FORTH a visszatérési vermen maradó, és az 1.-ba való visszatalálásra szolgáló cimet fogja felhasználni a 3. végén és a futás igy az 1. szóban fog folytatódni. Végül is a következőt kapjuk:

```
1. HARMADIK ELSO ok
```

Ugyanezt persze visszafelé is eljátszhatjuk: a visszatérési verem tetején levő cimet megduplázhatjuk. Emiatt a FORTH kétszer fog rátérni ugyanarra a felfűzött kódú ágra. Ha szavaink igy néznek ki:

programrészekre bizták. Mivel a konstansok és változók is szóként viselkednek (hiszen elfelejthetők, újradefiniálhatók stb), sejthető, hogy ezek is a szótárban vannak. De minthogy törzsük se nem gépi se nem felfűzött kód, hanem adat, a FORTH ezeket úgy különbözteti meg, hogy a DOCOL cime helyére a DOVAR, DOCON vagy DOUSE cimét irja.

A 6.13. ábra a változók cimének előállitását szolgáló DOVAR-t mutatja. A DSP regiszter az adatverem mutatója (Data Stack Pointer).

```
DOVAR:  W      <-- W+2
        DSP    <-- DSP-2
        (DSP)  <-- W
        JUMP NEXT
```

6.13. ábra
A DOVAR felépitése

Működése: a DOVAR meghivásakor a W-ben nyilván az a szótárcim van, ahol a DOVAR cime volt található. A FORTH a szótár következő memóriaszavát használja a változó értékének tárolására. Ezen üres hely cime nem más, mint W+2. A DOVAR ezt teszi az adatveremre a szokott módon, majd a program futása a NEXT-tel folytatódik.

A ó.14. ábrán a konstansok elővételét megoldó DOCON látható.

```
: 3. ." HARMADIK " R> DUP >R >R ;
: 2. 3. ." MASODIK " ;
: 1. 2. ." ELSO " ;
```

akkor az alábbi futtatási képet kapjuk:

1. HARMADIK MASODIK MASODIK ELSO ok

Végül a ;S hatását mutatjuk be. Tudjuk, hogy végrehajtásakor kerül az IP-be a visszatérési veremből az oda elmentett cim. Irjuk be a ;S-t a 2. szóba!

```
: 3. ." HARMADIK " ;
: 2. 3. ;S ." MASODIK " ;
: 1. 2. ." ELSO " ;
```

A 2.-ba irt ;S végrehajtása azonnali visszatérést eredményez az 1.-ba, és a MASODIK szöveg nem iródik ki (a ;S és ; közötti részből egy úgynevezett halott kód lett, hiszen arra sohasem kerül a vezérlés).

1. HARMADIK ELSO ok

Még egyszer megjegyezzük, hogy a fentieket illusztrációnak és nem követendő példának szántuk. A FORTH működését nem illik ily módon befolyásolni.

6.4.3. DOVAR, DOCON, DOUSE

Ebben a pontban a konstansok, változók és user változók belső kezelést mutatjuk be. A FORTH rendszer készitői mindhárom szerkezetet létrehozhatták volna a <BUILDS és DOES> segitségével, de az adatelővételi idő csökkentése céljából a megfelelő cimek és adatok veremre helyezését inkább speciális

6.4. Az utasitás-elővételi algoritmus

```
DOCON:  W      <-- W+2
        DSP    <-- DSP-2
        (DSP)  <-- (W)
        JUMP NEXT
```

6.14. ábra
A DOCON felépitése

A működés az előzőhöz hasonló. Amikor a NEXT rutin JUMP (W) utasitása hatására a DOCON-ra kerül a vezérlés, a W-ben az a szótárcim van, ahol, a DOCON cime volt. A W+2 cimen levő memóriaszót használja a FORTH a konstans értékének tárolására, innen veszi elő azt és helyezi az adatveremre.

A 6.15. ábrán láthatjuk a **DOUSE**-t. Ez a RAM-ban levő változók címének kiszámitását végzi.

```
DOUSE:  W      <-- W+2
        DSP    <-- DSP-2
        (DSP)  <-- UP+(W)
        JUMP NEXT
```

6.15. ábra
A DOUSE felépitése

Most a DOUSE cimét tartalmazó szótárrekesz utáni, W+2 cimen levő memóriaszóban a user változó sorszáma van. A RAM-ban ezen változók számára a UP (User Pointer = felhasználói mutató) cimtől kezdődően biztositott a hely. A DOUSE igy a UP-hez hozzáadja a (W)-en található sorszámot és az ily módon képzett cimet helyezi a veremre.

Ezzel befejeztük a FORTH futtatási algoritmusainak áttekintését. Ne feledjük, hogy a NEXT, DOCOL, DOVAR, DOCON

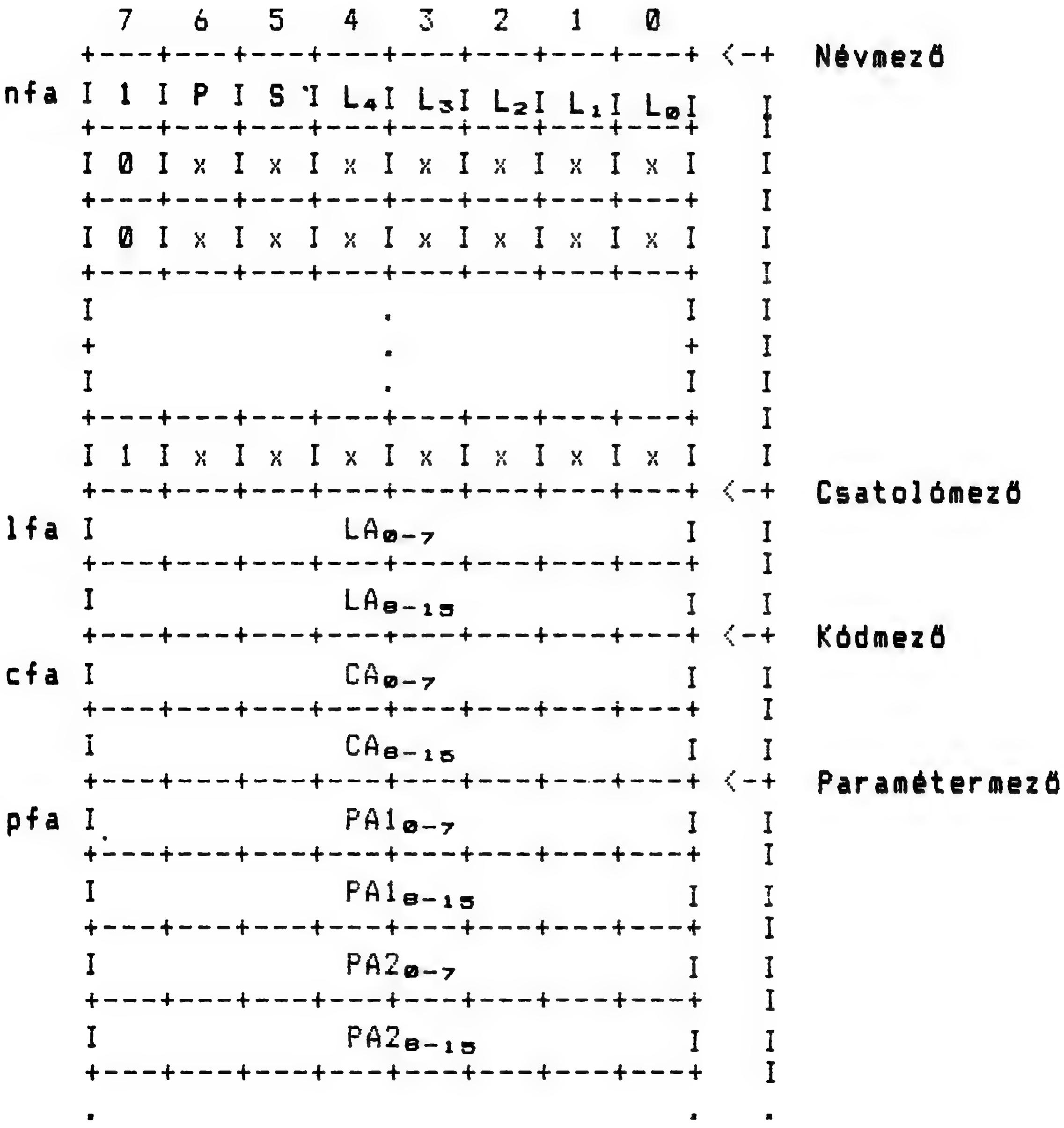
stb. egyike sem FORTH szó, hanem a futtatórendszer által használt alprogram! Egyes FORTH változatok lehetővé teszik a NEXT, IP és W programból való elérését, de ezeket is legfőképp gépi kódú szavakban használják. A futtatórendszer rutinjainak használata csak a FORTH működésének alapos ismerete és kellő körültekintés mellett javasolható.

6.5. A szó belső ábrázolása

A 3. fejezetben nagyvonalakban már szóltunk a szavak belső ábrázolásáról. Ott a szótári bejegyzést fejre és törzsre bontottuk. Ezt a durva felbontást most tovább finomitjuk, és részletesen megismerkedünk a bejegyzések szerkezetével. Végül a szó belső szerkezeti elemeinek elérésére szolgáló utasitásokat mutatjuk be.

6.5.1. Az nfa, lfa, cfa, pfa fogalma és használata

A 6.16. ábrán láthatjuk a szótári bejegyzések általános felépitését.



ó.16. ábra
Egy szótári bejegyzés felépitése

6.5. A szó belső ábrázolása

A bejegyzés első eleme a **névmező**. Azt a cimet, ahonnan ez kezdődik, **nfa**-nak nevezzük (name field address = névmezőcim). A névmező első bájtja a **státuszbájt**. Ennek alsó öt bitje (L_0-L_4) tartalmazza a szó nevének hosszát. Emiatt a szónév nyilvánvalóan legfeljebb 31 karakterből állhat, mint azt már emlitettük.

Az ötödik bit a smudge bit (S). Erről a 3. fejezetben a rekurzió tárgyalásánál már volt szó. Ha S=1, a FORTH az adott szót a szótárban való keresés során figyelmen kivül hagyja, ismeretlennek, definiálatlannak tételezi fel. Egy szó forditása közben a smudge bit mindig 1, kivéve azt az esetet, amikor ezt a SMUDGE szó segitségével megváltoztatjuk.

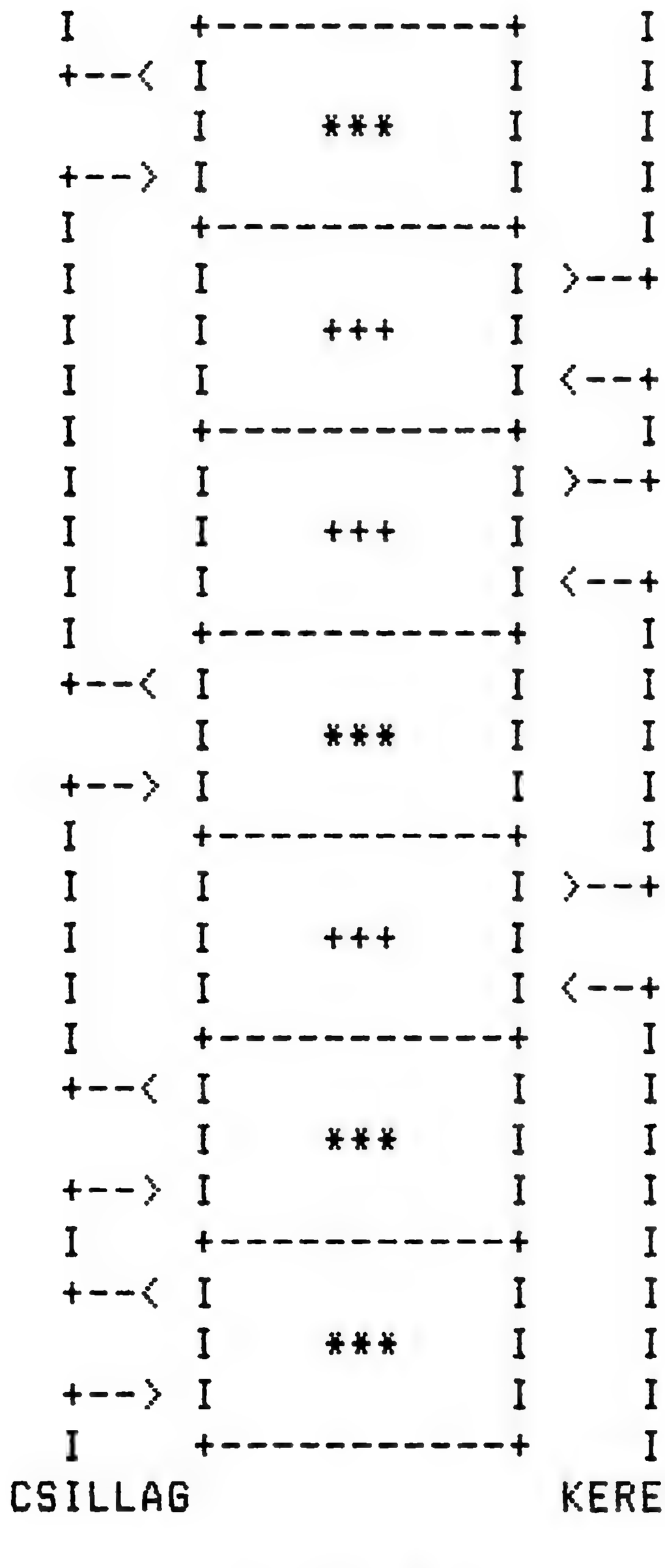
A 6. bit a precedenciabit (P). Ez a bit mondja meg, hogy az adott szótári bejegyzés közvetlen szóra vonatkozik-e. Ha P=1, akkor a szó közvetlen. A 7. bit mindig 1.

Ezután következik a névmezőben a szó neve, amely ASCII karakterek formájában kerül tárolásra. Az ASCII karakterkészletben a kódok 0 és 127 között vannak. Igy a 7. bit mindig 0, kivéve a név utolsó karakterét, ahol a 7. bit egyesre van állitva, jelezve, hogy ez a névmező utolsó bájtja.

A bejegyzés következő eleme a **csatolómező**. A csatolómező két bájt, és cimét **lfa**-nak nevezzük (link field address = csatolómezőcim). A szótárban való kereséskor a FORTH-nak sorban végig kell vizsgálnia az egyes bejegyzések névmezőiben tárolt neveket a keresettel, hogy az adott szót azonositani tudja. Mivel a bejegyzések hossza változó, szükséges minden bejegyzésben valamilyen információt tárolni, amely alapján a soron következő megtalálható. Ez az információ a **csatolási cim** (**LA** = Linking Address), amely a csatolómezőben található. A csatolási cim nem más, mint az előző bejegyzés nfa-ja. Tehát ha először az EGYIK, majd a

MASIK nevű szót definiáljuk, akkor a MASIK csatolómezőjében az EGYIK nfa-ja, az EGYIK csatolómezőjében az előtte definiált szó nfa-ja van és igy tovább. Természetesen a láncolás az alapszótárra is igaz. A felhasználói szótár első szava mutat az alapszótár utolsó szavára. Az alapszótár legelső szavának csatolómezőjében 0 van, jelezve, hogy ez a szótár vége. (Elsőre talán meglepőnek tűnik, hogy a szótár a nagyobb memóriacimek felé növekszik, de az eleje mindig a legutoljára definiált szó, a vége az alapszótár első szava és a csatolás is a csökkenő cimek felé történik. Ennek magyarázatára a 6.5.2. pontban visszatérünk).

A csatolómező nemcsak a szótárban való keresést könnyiti meg, hanem lehetővé teszi az egyazon szótárba tartozó szavak összekapcsolását is. Az 5. fejezetben már láttuk, hogy több szótárat is létrehozhatunk és a szódefiniciókat tetszőlegesen helyezhetjük el a megfelelő szótárban. A különböző szótárakhoz tartozó bejegyzések azonban fizikailag nem különülnek el, hanem a memóriában folytonosan helyezkednek el. Ez a tár gazdaságosabb kihasználását eredményezi. Az egyes szótárakat a csatolómező segitségével különbözteti meg a FORTH, mint ahogy az a 6.17. ábrán látható.



6.17. ábra
Két szótár megkülönböztetése a
csatolómezők segitségével

Az ábrán minden téglalap egy bejegyzés. A *** jelűek a CSILLAG, a +++ jelűek a KERESZT nevű szótárhoz tartoznak. Megfigyelhetjük, hogy a megfelelő láncolás miatt a fizikailag el nem különülő (azaz keveredő) *** és +++ jelű szavak két, logikailag különálló szótárra válnak szét. Nyilvánvaló, hogy ez a módszer tetszőlegesen sok szótár szétválasztására is alkalmas.

A bejegyzés csatolómezőt követő eleme a **kódmező**, amelynek címét **cfa**-nak nevezzük (**c**ode **f**ield **a**ddress = kódmezőcim). Könyvünk előző részeiben azon, hogy a "szó cime", mindig a cfa-t értettük.

A felfűzött kódban szereplő cimek is kódmezőcimek, a végrehajtandó szavak cfa-i. A NEXT során a W-be tehát a cfa kerül, igy a NEXT mindig a kódmezőben levő **kódcimre** (CA = Code Address) adja a vezérlést. Hogy ez a kódcim mi lehet, azt a 6.4. szakasz alapján könnyen kitalálhatjuk: vagy a gépi kódra mutató cim, vagy a DOCOL, DOVAR stb. cime.

A bejegyzés utolsó eleme a **paramétermező**. A paramétermező bossza szavanként változik, és a következő szó névmezejéig tart. A mező kezdőcimét **pfa**-nak hivjuk (**p**arameter **f**ield **a**ddress = paramétermező-cim). A paramétermező az adott szó természetétől függően mást és mást tartalmazhat. Gépi kódú szó esetén itt található a gépi kódú program, amelynek kezdőcime igy a pfa. Ilyenkor a kódmezőben a pfa van mint ugrási cim (l. a 6.4.1. pontot). Ha a szó felfűzött kódú, akkor a paramétermezőben a felfűzött kód van. Ekkor a kódmezőben a DOCOL cime helyezkedik el, és az IP a visszatérési verembe való mentés után a pfa-t fogja tartalmazni. Végül konstansok, változók, <BUILDS és DOES> segitségével létrehozott egyéb adatszerkezetek esetén a paramétermező adatokat tartalmaz, és ezek elővételét a kódmezőben levő cimen kezdődő programrészlet végzi (például a DOCON vagy a DOVAR).

6.5.2. Utasitások a szó belső elemeinek elérésére

Először a bejegyzés négy meghatározó címének, az nfa, lfa, cfa és pfa-nak a meghatározásával foglalkozunk. Az ' (aposztrof), (--- pfa) szó szolgál arra, hogy egy szó nevéből annak pfa-ját megkapjuk. Az közvetlen szó. Ezért szódefinició belsejében, ha azt akarjuk, hogy az ' csak az adott szó végrehajtásakor jusson érvényre, előtte a [COMPILE]-t kell alkalmaznunk. (Erre példákat is fogunk látni). Irassuk ki például az ABS szó paramétermező-cimét.

```
HEX ok
  ABS U. 9178 ok
```

A HEX szót azért használjuk, mert a cimeket hexadecimális számrendszerben szokták kiirni.

A pfa ismeretében a többi cimet is könnyedén meghatározhatjuk az úgynevezett cimkonverziós szavak segitségével. A **CFA** (pfa --- cfa) szó a kódmezőcimet, az **NFA** (pfa --- nfa) a névmezőcimet és végül az **LFA** (pfa --- lfa) a csatolómező-cimet adja meg. Mindháron szó a paramétermező-cimet számitja át cfa-vá, nfa-vá vagy lfa-vá. Jegyezzük meg, hogy a legutoljára definiált szó nfa cimét megadja a **LATEST** szó (--- nfa). Abban az esetben, ha egy szó névmezőcimét ismerjük (például a cimet az őt megelőző szó csatolómezőjéből olvastuk ki), akkor ebből a **PFA** (nfa --- pfa) szó felhasználásával meghatározhatjuk az adott névmezőcimhez tartozó paramétermező-cimet. A pfa ismeretében viszont már a cfa és lfa egyszerűen számitható. A 6.18. ábra összefoglalja az eddig ismertetett szavakat.

```
' név          --- pfa
NFA     pfa --- nfa
LFA     pfa --- lfa
CFA     pfa --- cfa
PFA     nfa --- pfa
LATEST      --- nfa
```

6.18. ábra
Címkonverziós szavak

Következő példánk CIMET.IR szavát felhasználhatjuk arra, hogy egy tetszőleges szóhoz tartozó bejegyzés négy meghatározó címét kiirassuk.

```
: CIMET.SZAMOL ( --- pfa cfa lfa nfa )
                [COMPILE] ' DUP DUP DUP
                CFA ROT LFA ROT NFA ; ok
: CIMET.IR CIMET.SZAMOL
           CR ." NFA=" U.
           CR ." LFA=" U.
           CR ." CFA=" U.
           CR ." PFA=" U. ; ok
HEX ok
CIMET.IR MAX
NFA=91A5
LFA=91A9
CFA=91AB
PFA=91AD ok
```

Figyeljük meg a [COMPILE] alkalmazását a CIMET.SZAMOL szóban! Ha ez nem lenne ott az ' előtt, akkor az '-ot követő DUP paramétermezőcime kerülne a veremre a CIMET.SZAMOL definiciója közben. Igy viszont az csak a CIMET.IR végrehajtásakor jut érvényre, tehát bemenő paramétere a MAX lesz.

Irjunk most egy olyan szót, amely egy bejegyzés státuszbájtja alapján eldönti, hogy közvetlen vagy nem közvetlen szóhoz tartozik-e, és ezt ki is irja.

```
: P.BIT? ( pfa --- p.bit )
         [COMPILE]   NFA C@ 64 AND ; ok
: KOZVETLEN? P.BIT? IF ." KOZVETLEN "
                    ELSE ." NEM KOZVETLEN "
                    ENDIF ; ok
KOZVETLEN? + NEM KOZVETLEN ok
KOZVETLEN? IF KOZVETLEN ok
```

A P.BIT? szóban először kiszámitjuk az által adott paramétermező cimből a bejegyzés névmezőcimét. A névmező első bájtja a státuszbájt. Ebben a hatodik bit a precedenciabit, amit a 64-gyel (bináris 01000000) kimaszkolunk. A keletkező nulla vagy nemnulla érték közvetlenül felhasználható a KOZVETLEN? szóban levő feltételes elágazás vezérléséhez.

Ha szónévből cimet tudunk számitani, akkor feltehetőleg van olyan szó is, amely cimből szónevet állit elő. Valóban, erre való az **ID.** szó. A verem tetején levő névmezőcimhez tartozó szónevet irja ki. Készitsünk például egy ELOZO nevű szót, amely kiirja azt, hogy egy megadott szó csatolómezője melyik szóra mutat.

```
: ELOZO [COMPILE] ' LFA @ ID. ; ok
ELOZO FORGET ' ok
ELOZO DABS ABS ok
```

Abban az esetben, ha egy szó névmezője egyik végének cimét ismerjük és a másik vég cimére vagyunk kiváncsiak, a **TRAVERSE** szót kell használnunk:

```
TRAVERSE     cim1 n --- cim2
```

szótárméretet elérve már egyre inkább a saját szavainkat használjuk, és egyre kevésbé az alapszótár szavait. Mivel a szótár eleje mindig a legutoljára definiált szó, és a keresés innen kezdődik, a saját szavak megtalálásának ideje lerövidül.

A FIND beolvassa és a DP aktuális értékétől kezdve tárolja a keresendő szó nevét. Ezután elkezdi a szótárban a szó keresését. Először a CONTEXT, majd ha eddig sikertelen volt, akkor a CURRENT által kijelölt szótárban történik a keresés. Ha megtalálta akkor a veremre teszi a szó kódmezőjének cimét. Ha a keresés sikertelen volt, akkor csak egy logikai hamis érték (tehát nulla) kerül a veremre:

```
FIND        --- cfa
            --- 0
```

Kerestessük meg például a DUP szót a FIND-dal! A veremre tett cimet irassuk ki!

```
HEX ok
FIND DUP U. 84F2 ok
```

A fig-FORTH a FIND helyett a -FIND szót biztositja (--- pfa st j vagy j). Feladata azonos a FIND-éval, eltérés a kimenő paraméterekben van. A -FIND az őt követő szó nevét beolvassa, és a DP értékétől kezdve tárolja. Ezután elkezdi a szó keresését. Ha megtalálta, a verem tetején elhelyezi annak pfa cimét, státuszbájtját (st), és egy logikai igaz értéket (j=-1). Sikertelen keresés esetén csak egy logikai hamis értéket hagy a vermen (j=0).

Ha a szó nevének beolvasásáról magunk akarunk gondoskodni, valamint azt is meg akarjuk adni, hogy a keresés melyik szótól kezdődjön, akkor a (FIND) szót használjuk(ciml cim2 --- pfa st j vagy j). Ennek két bemenő

Cim1 a névmező egyik végének cime. Ez lehet vagy maga az nfa, azaz a státuszbájt cime, vagy lehet a név utolsó karakterének cime. Az n mutatja azt, hogy a FORTH merre keresse a másik véget. Ha n=1, akkor cim1 az nfa és az eredményül kapott cim2 az utolsó karakter cime lesz. Az n=-1 esetben cim1-nek az utolsó karakter cimének kell lennie, cim2 pedig az adott szó nfa-ja lesz. Egy szó névmezőcime ismeretében például annak csatolómező-cimét kétféle módon is meghatározhatjuk. Tegyük fel, hogy a vermen a MAX szó nfa-ja, 91A5 van. Az első változat a TRAVERSE nélkül:

PFA LFA U. 91AB ok

Ha a TRAVERSE-t használjuk, akkor a cim fölé a veremre egyet kell tennünk, jelezve, hogy az nfa-t ismerjük. Az eredményül kapott cimhez egyet hozzáadva kapjuk az lfa-t (hogy miért, az a 6.16. ábrából látható).

1 TRAVERSE 1+ U. 91AB ok

Mindkét változatot alkalmazhatjuk, bár megjegyzendő, hogy a TRAVERSE-t nem lehet igazából kikerülni, mert a PFA szó is felhasználja a TRAVERSE-t az nfa pfa-vá konvertálásához.

Végül ismerkedjünk meg két kereső utasitással! A szavakat a FORTH használja compiler üzemmódban a szótörzset alkotó szavak felimerésére és paramétermező-cimük meghatározására. A szavak alkalmazásának akkor lehet jelentősége, ha ismeretlen szó esetén a hiba kezelését magunk akarjuk megoldani.

Az első kereső szó a **FIND** (= megtalálni), (--- cfa vagy 0). Jó tudni, hogy a szótárban a keresés a csökkenő cimek felé történik. Ennek magyarázata egyszerű. Gondoljuk meg, hogy a programfejlesztés során egy bizonyos

paramétere van. A verem tetején annak a szónak a nevmezőcimét várja, amely szótól a keresést el kell kezdeni. Az ez alatt levő veremszinten pedig azt a cimet kell megadni, ahol a keresendő szó neve található. A (FIND) kimenő paramétere a szó paramétermező-cime, a státuszbájtja és egy logikai igaz érték (j=-1), ha a keresés sikeres volt. Ellenkező esetben csak egy hamis értéket kapunk (j=0):

```
(FIND)          cim1 cim2 --- pfa st igaz
                cim1 cim2 --- hamis
```

A FIND és a -FIND is a (FIND)-ot használja a szöveg beolvasása és a paraméterek beállitása után. Keresésre egyszerűbb volta miatt általában a FIND-ot vagy a -FIND-ot alkalmazzuk.

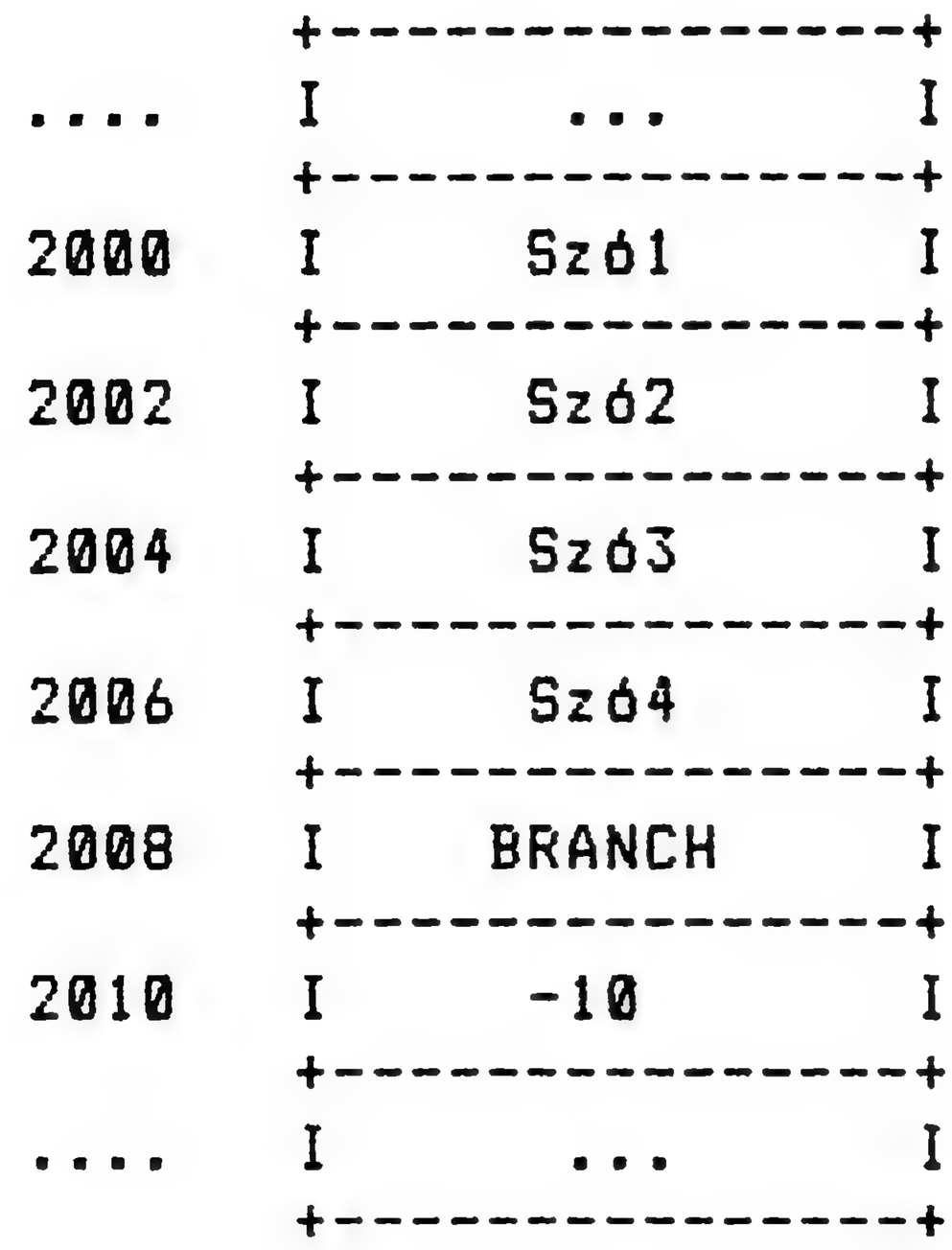
6.6. Vezérlésátadás a felfűzött kódban

Ebben a szakaszban bemutatjuk, hogy miként valósitható meg a felfűzött kódban a vezérlésátadás. Valóságos processzorok esetén a vezérlésátadás a programszámláló megváltoztatását jelenti. Ezzel a program futása nem a sorban következő utasitással folytatódik, hanem egy másik, általunk kijelölt pontról. A programszámláló értékét kétféleképpen lehet megváltoztatni. Az első esetben a processzor kiolvassa a memóriából az ugrási cimet, és ezt a PC-be tölti. A másik lehetőség, hogy a memóriában nem az ugrási cim van, hanem egy olyan érték, amelyet az aktuális program-számláló-tartalomhoz hozzáadva megkapjuk a folytatási cimet. (Ha ez a szám pozitiv, akkor előre, ha negativ, akkor visszafelé ugrás történik a programban).

A FORTH ezen második tipusú vezérlésátadást alkalmazza. A programszámláló szerepét itt, mint tudjuk, az IP regiszter

veszi át. Az első szó, amivel megismerkedünk, a **BRANCH** (= ugorj). A szótárban a BRANCH cime utáni memóriaszóban kell elhelyezni az ugráshoz szükséges eltolást. A működési mechanizmus a következő: amikor a BRANCH-re kerül a vezérlés, akkor mint tudjuk, az IP-ben az a cim van, ahol a BRANCH cime volt. A BRANCH hozzáadja az IP-hez az őt követő eltolást, és ezzel az IP most a felfűzött kód egy más pontjára fog mutatni. A BRANCH-et követő NEXT az IP-t kettővel növeli, és ahova az most mutat, az lesz az ugrás után végrehajtott első szó. Tekintsük most a 6.19. ábrát! (A könnyebb számolás érdekében most és a továbbiakban is a cimek és az eltolások is decimális számként értelmezendők).

6.6. Vezérlésátadás a felfűzött kódban



6.19. ábra
Ugrás a felfűzött kódban

Amikor a BRANCH végrehajtására kerül a sor, az IP-ben 2008 van. A BRANCH az őt követő memóriaszóban levő eltolást, a -10-et adja hozzá az IP-hez, igy az 2008-10=1998 lesz. Az ezután végrehajtandó NEXT az IP-t kettővel növeli, tehát az 2000-t fog tartalmazni. A program futása a Szó1-gyel folytatódik. Az ábrán tehát egy végtelen ciklust láthatunk, amelynek magja a Szó1 ... Szó4. Általánosan annak a szónak a cimét, amellyel a futás a BRANCH végrehajtása után folytatódik, igy kaphatjuk meg:

⟨folytatási cim⟩ = ⟨BRANCH cime⟩ + ⟨eltolás⟩ + 2

A FORTH egy feltételes ugrásutasitással is rendelkezik, ez a **0BRANCH**. Működése egyetlenegy ponton tér el a

BRANCH-től: az ugrás csak akkor hajtódik végre, ha a verem tetején nulla van. Ellenkező esetben a felfűzött kód végrehajtása a 0BRANCH-et követő eltolás utáni szóval folytatódik. Mindkét esetben törlődik a verem legfelső eleme.

A következő pont az eltolás kiszámitását megkönnyitő szavakkal és a BRANCH, ill. 0BRANCH használatával ismertet meg.

6.6.1. Ugrási eltolás számitása a BACK szóval

A gyakorlatban ritkán van szükségünk arra, hogy a program vezérlését a BRANCH vagy a 0BRANCH szavakkal oldjuk meg. Ezt ugyanis a FORTH magas szintű vezérlési szerkezeteivel sokkal könnyebben valósithatjuk meg. Ha viszont egy új szerkezetet szeretnénk létrehozni, akkor esetleg szükségünk lehet a BRANCH-ek használatára.

Lássuk, miként jön létre például egy, a 6.19. ábrán látható szótárrészlethez hasonló struktúra. Beláttuk, hogy az egy végtelen ciklus. Ilyet a FORTH-ban a BEGIN és AGAIN segitségével lehet létrehozni. Nézzük először a BEGIN és AGAIN definicióját, majd a 3. fejezetben már ismertetett VEGTELEN szót.

```
: BEGIN ?COMP HERE 1 ; IMMEDIATE
: AGAIN 1 ?PAIRS COMPILE BRANCH BACK ; IMMEDIATE

: VEGTELEN BEGIN KEY EMIT AGAIN ;
```

Kövessük most végig, mi történik a VEGTELEN forditása közben, valamint ismerkedjünk meg az eddig ismeretlen

szavakkal! Tegyük fel, hogy a FORTH már elkészitette a VEGTELEN fejét, kitöltötte névmezőjét, a kódmezőbe beirta a DOCOL cimét. A DP most a leendő paramétermezőre mutat, ide kerülnek majd a szótörzs szavainak kódmezőcimei. Legyen a DP-ben mondjuk 4000. Az első szó a BEGIN, amely közvetlen, tehát azonnal végrehajtódik. A ?COMP szó arra szolgál, hogy megakadályozza a BEGIN interpreter állapotban való leirását. (A 3. fejezetben már emlitettük, hogy a vezérlési szerkezetek csak szódefinició belsejében használhatók). A ?COMP forditás állapotban semmit nem csinál, de interpreter módban hibajelzést ad. A szó ellentettje a ?EXEC. Ez akkor jelez hibát, ha végrehajtására forditás közben kerül sor.

Mivel most szódefinició közben vagyunk, a ?COMP nem jelez hibát, és végrehajtódik a HERE szó. Ezzel a veremre kerül a DP értéke, a 4000. Ezután az egyes iródik a veremre, amelynek szerepére az AGAIN-nél derül fény. A BEGIN végrehajtása ezzel befejeződött és sor kerül az őt követő szavak feldolgozására. Sem a KEY, sem az EMIT nem közvetlen, igy ezek kódmezőcimei a szótárba kerülnek. A KEY cfa-ja a 4000-es, az EMIT-é a 4002-es cimre kerül és a DP-ben 4004 lesz, hiszen ez a soron következő üres bájt cime. A VEGTELEN következő szava az AGAIN, a BEGIN-hez hasonlóan szintén közvetlen szó. Lássuk, végrehajtása mit eredményez!

Elsőnek egy egyes kerül a veremre. Most két egyes van a vermen, alattuk pedig a 4000, a DP értéke a BEGIN végrehajtásakor. Ezt követően hivódik meg a ?PAIRS (= ?párok) szó. Ez megvizsgálja, hogy a verem tetején levő két szám egyenlő-e. Ha igen, akkor semmi sem történik, ha viszont nem azok, akkor a ?PAIRS egy hibajelzést generál. A két számot persze mindkét esetben leveszi a veremről. Igy tudja a FORTH megvizsgálni, hogy egy vezérlési szerkezetnek van-e eleje. " Itt vagyok! " - jelzi a BEGIN a maga 1-évei. " Lássuk, volt-e BEGIN? " - kérdezi az AGAIN az 1-el és a ?PAIRS-el. Persze kérdés az is, hogy mi van akkor, ha volt

BEGIN, de elfelejtettük leirni az AGAIN-t. Ekkor sincs baj, mert igy meg a BEGIN egyesét semmi sem használja fel, a ; elérésekor tehát a veremmutató nem egyezik meg a CSP-ben tárolt értékkel, ezért most is hibajelzést kapunk. A leirt ellenőrzési eljárást használja a többi vezérlési szerkezet is, persze más-más számokkal. Azt nem kell magyarázni, hogy ez a módszer miért nem nyújt száz százalékos védelmet a csonka szerkezetek ellen, de az esetek nagy többségében ez is kielégitő biztonsággal jelzi a hibákat.

E kitérő után foglalkozzunk megint az AGAIN-nel. A ?PAIRS nem talált hibát, a két egyest levette a veremről, igy ott csak a 4000 maradt. Most jön a COMPILE, ami beforditja a szótárba az utána álló szó cimét. Igy a BRANCH kódmezőcime bekerül a szótár következő üres memóriaszavába, a 4004-be, a DP-ben pedig ezután 4006 lesz. Ezután kerül sor a **BACK** (= vissza) szóra. A BACK definiciója igen egyszerű:

```
: BACK HERE - , ;
```

(Ez persze rendszerszó, nem nekünk kell definiálni!) A BACK a veremre teszi a DP aktuális értékét, ami most 4006. Ezután a BEGIN által a veremre irt 4000-ből kivonja a 4006-ot, és az eredményt, a -6-ot beirja a szótár következő üres szavába. Ezt a DP jelöli ki, és most 4006. Ez a -6 lesz a BRANCH működéséhez szükséges ugrási eltolás. Ezzel az AGAIN végrehajtódott, a szótárba kerül még a ;S cime, és igy a VEGTELEN definiálása lezárul. A szótár egy részlete látható a 6.20. ábrán.

6.6. Vezérlésátadás a felfűzött kódban

```
          +----------------+
....      I      ...       I
          +----------------+
4000      I      KEY       I
          +----------------+
4002      I      EMIT      I
          +----------------+
4004      I    BRANCH      I
          +----------------+
4006      I       -6       I
          +----------------+
4008      I       ;S       I
          +----------------+
....      I      ...       I
          +----------------+
```

6.20. ábra
A VEGTELEN paramétermezője

Nézzük meg a már ismertetett képlet alkalmazásával, hogy a -6 ugrási eltolás hatására a BRANCH tényleg a KEY-re adja-e a vezérlést.

⟨folytatási cim⟩ = ⟨BRANCH cime⟩ + ⟨eltolás⟩ + 2 =
= ⟨4004⟩ + ⟨-6⟩ + 2 =
= ⟨4000⟩

A folytatási cim tehát 4000, ami valóban a KEY cime. Általánosan azt mondhatjuk, hogy az ugrási eltolásnak a BACK szóval való kiszámitása esetén az alábbiak szerint kell eljárni: először a veremre kell tenni azt a szótárcimet, ahova majd ugrani akarunk a BRANCH vagy ØBRANCH segitségével. Ez általában a HERE szó végrehajtásával történik azelőtt, mielőtt az ugrás céljául szolgáló szó a szótárba kerül. Ezután tetszőleges műveletek következnek, további szavak

kerülnek a szótárba, stb. Végül az ugrás helyén a szótárba forditunk egy BRANCH vagy ØBRANCH utasitást, majd végrehajtunk egy BACK-et. Ennek hatására a megfelelő ugrási eltolás automatikusan a szótárba kerül.

A BACK csak a szótárban visszafelé, az alacsonyabb cimek felé történő ugrások megvalósitását segiti elő. Előrefele történő ugrások kiszámitása sem sokkal bonyolultabb. Itt egyedül az jelent problémát, hogy tudjuk az ugrási eltolás helyét, de értékét még nem ismerjük. Gondoljuk meg: példának okáért az IF úgy működik, hogy egy feltételes ugrást fordit a szótárba, amely az IF-et vezérlő feltétel hamis volta esetén az ENDIF utáni szóra adja a vezérlést. (Tegyük fel, hogy ELSE ág most nincs). Igen ám, de az IF még nem tudhatja, hogy milyen messze lesz az ENDIF. Ezért az IF kihagyja a ØBRANCH után az ugrási eltolás helyét (pontosabban egy nullát ir oda), és majd csak az ENDIF számolja ki az eltolás értékét és irja be a ØBRANCH utáni memóriaszóba. Ehhez az eltolás cimét az IF teszi a veremre.

További magyarázatok helyett közöljük az IF és ENDIF definiciójáť, valamint az ELOJELE nevű szót, amely egy minuszjelet ir ki, ha a vermen egy negativ szám van. Kérjük az Olvasót, hogy a definiciók alapján kövesse végig lépésenként az ELOJELE forditását úgy, ahogy ezt a VEGTELEN esetében tettük. Ily módon lássa be, hogy az ELOJELE paramétermezője olyan lesz, mint ahogy az a 6.21. ábrán látható! Tételezze fel, hogy az ELOJELE pfa-ja 5000. Ha a forditás menetét gondosan végigköveti, választ kaphat arra, miképpen történik az előreugrások paramétereinek kiszámitása.

```
: IF COMPILE 0BRANCH HERE 0 , 2 ; IMMEDIATE
: ENDIF ?COMP 2 ?PAIRS
        HERE OVER - SWAP ! ; IMMEDIATE

: ELOJELE 0< IF 45 EMIT ENDIF ;
```

5000	0<
5002	0BRANCH
5004	+8
5006	LIT
5008	45
5010	EMIT
5012	;S

6.21. ábra
Az ELOJELE paramétermezője

6.6.2. EXECUTE

Előfordulhat, hogy egy szót úgy kell végrehajtanunk, hogy csak a pfa-ja ismert. Például valamilyen feltételtől függően más és más szavakat kell végrehajtani, a szavak pfa-i pedig egy ugrási táblázatban találhatók. Ilyenkor az **EXECUTE** (= hajtsd végre), (pfa ---) szót használjuk. Az EXECUTE működését az alábbi példán láthatjuk:

6.6. Vezérlésátadás a felfűzött kódban

```
: HAHO ." HAHO " ;
```

Ezt természetesen végrehajthatjuk a HAHO leirásával, de tehetjük a következőt is:

```
  HAHO  ( --- pfa ) ok
EXECUTE ( pfa --- ) HAHO ok
```

A következő példa már használhatóbb. Lássuk, hogyan oldhatjuk meg, hogy a HAHO szó hol azt irja ki, hogy "jó reggelt", hol meg azt, hogy "jó estét":

```
: REGGEL ( --- )
         ." jó reggelt " ; ok
: ESTE   ( --- )
         ." jó estét " ; ok
VARIABLE 'HAHO ok
: HAHO   ( --- )
         'HAHO @  ( --- pfa )
                  EXECUTE ( pfa --- )
; ok
' REGGEL ( --- pfa ) 'HAHO ! ok
HAHO jó reggelt ok
```

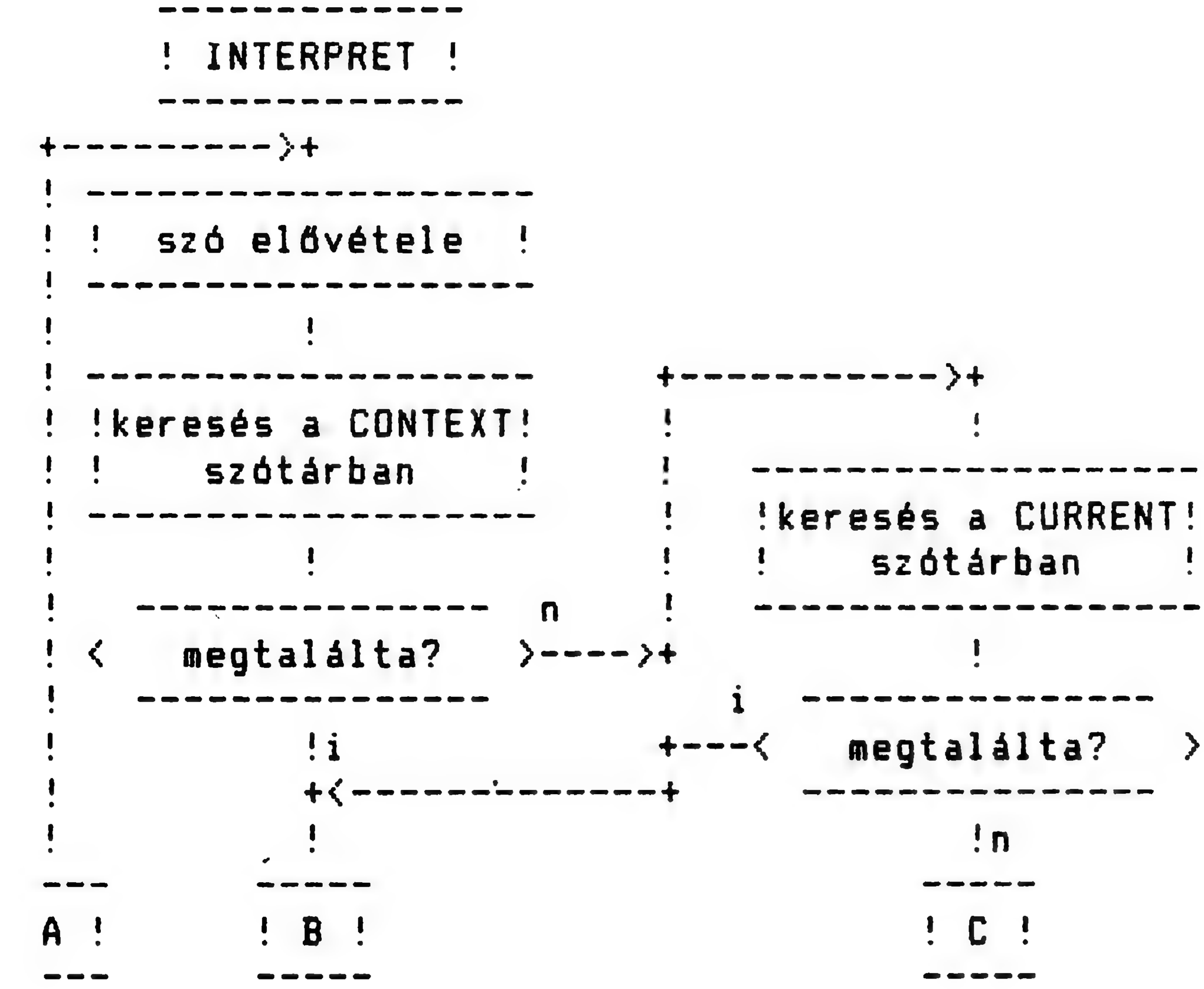
Ha most a másik üzenetet akarjuk kiiratni, a következőt tesszük:

```
  ESTE 'HAHO ! ok
HAHO jó estét ok
```

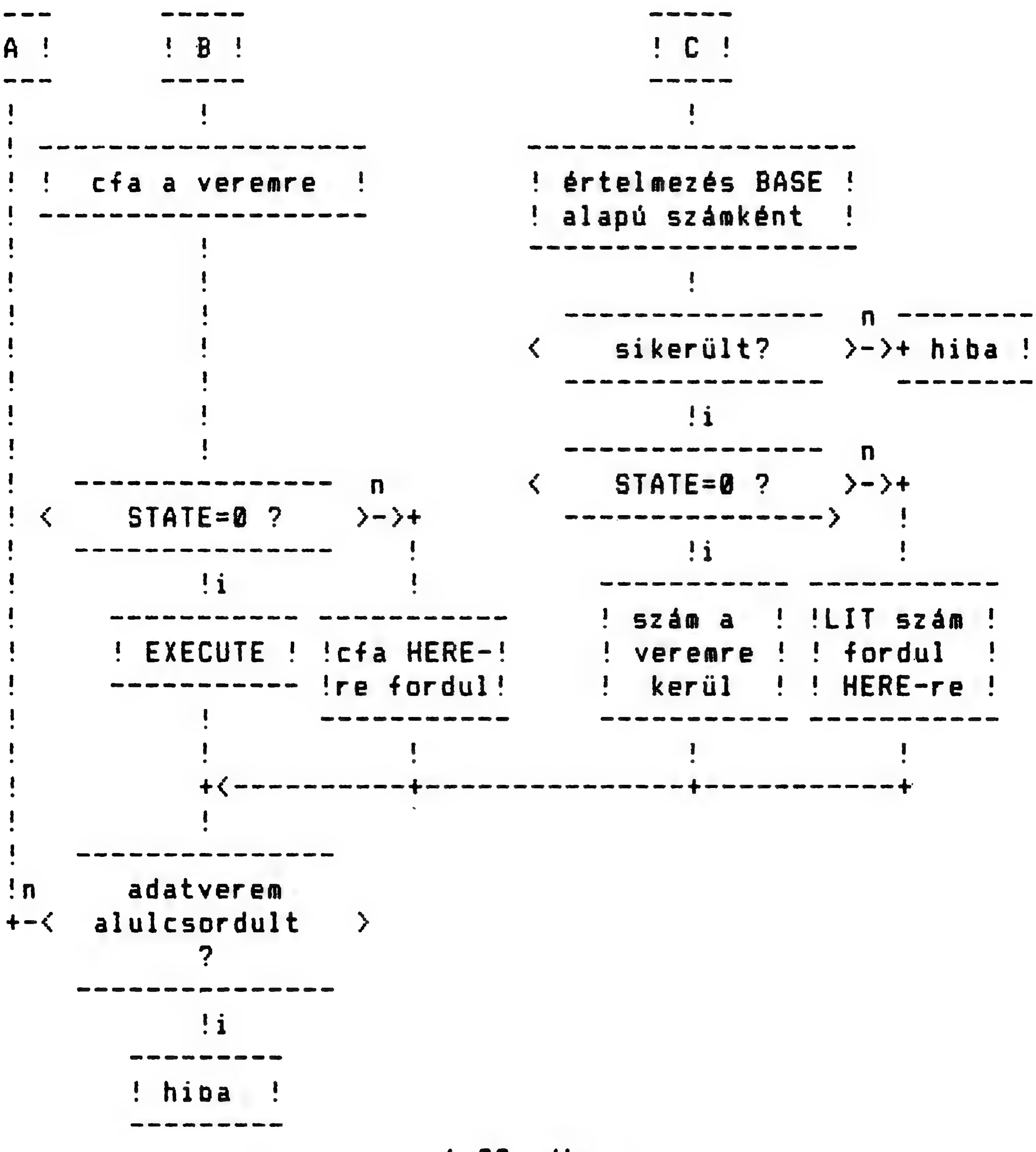
A példa önmagáért beszél, megértését ezért az Olvasóra bizzuk.

6.6.3. INTERPRET

Eddig megismertük a FORTH utasitás-végrehajtásának algoritmusát. Ebben a pontban a szöveginterpreterrel foglalkozunk. Ez dolgozza fel az inputfolyamot, ismeri fel a szavakat, számokat, ezeket a STATE-től függően végrehajtja vagy a szótárban helyezi el. Ezt a feladatot ellátó szó neve **INTERPRET** (---), (az "interpret" szó jelentése: értelmezd, tolmácsold). Az interpretálandó szöveg kezdőcimére az IN változó mutat. úgy véljük, hogy hosszas magyarázkodásnál sokkal érthetőbb egy folyamatábra. Az alábbi folyamatábra alapján az INTERPRET működése könnyen megérthető.



6.6. Vezérlésátadás a felfőzött kódban



6.22. ábra
Az INTERPRET működésének folyamatábrája

6.6. Vezérlésátadás a felfűzött kódban

Az INTERPRET-tel nem foglalkozunk tovább, mert a programozói gyakorlatban csak ritkán fordul elő, főképp rendszerprogramokban használatos. Helyette gyakoribb az EXECUTE használata.

6.7. Rendszerváltozók és rendszerkonstansok

Ezt a szakaszt a rendszerváltozók és rendszerkonstansok összefoglalására szántuk. Közöljük azon változók leirását is, amelyekről esetleg már könyvünk előző részeiben is szóltunk. Egyes változók leirásánál a könyvben eddig nem ismertetett, új szavakat is bemutatunk. Mivel a felsorolt szavak működése vagy kézenfekvő, vagy csak igen kis jelentőségű, de főleg azért, mert a rendszer használja őket, a szavak használatát nem mutatjuk be példákon keresztül.

6.7.1. Rendszerváltozók

BASE
A FORTH által használt számrendszer alapszáma. Számkiiratáskor e számrendszer alapján iródnak ki a számok, beolvasáskor pedig úgyszintén BASE alapján történik a számkonverzió. Állitható egyrészt a DECIMAL ill. HEX szavakkal 10-re vagy 16-ra, másrészt az " n BASE ! " utasitásokkal egyéb értékekre. Az n legfeljebb 36 lehet.

BLK
A változó nulla értéke esetén a FORTH által feldolgozandó szódefiniciók a terminálról érkeznek. Ha viszont a fordítás blokkokból történik, akkor BLK az éppen fordítás alatt álló blokkra mutat.

CSP
Ezt a változót a FORTH rendszer használja az adatveremmutató átmeneti tárolására. A veremmutató CSP-be való irására külön szó szolgál, ez a !CSP. A FORTH megköveteli, hogy egy szódefinició végén a verem mélysége megegyezzen a szódefinició kezdetén fennálló mélységgel. Tehát : és ; között amit a veremre teszünk, azt le is kell venni onnan. Ezért a : szó végrehajt egy !CSP utasitást, a ; pedig egy ?CSP-t. A ?CSP szó összehasonlitja a veremmutató aktuális értékét a CSP-ben tárolttal, és ha a kettő nem egyezik meg, akkor egy hibaüzenetet generál. A változót és a hozzá tartozó két szót magunk is használhatjuk, de szódefinició közben ekkor a CSP tartalmát el kell menteni és a ; előtt vissza kell állitani.

CONTEXT
Az aktuális keresési szótár csatolómezőjének cimét tartalmazza. Ezen csatolómező mutat a szótár első szavára. A szótárbeli keresés mindig a CONTEXT által kijelölt szótár első szavával kezdődik. Magát a szó nevét a CONTEXT @ @ ID. utasitásokkal irathatjuk ki. Alapértelmezésben a változóban levő cimen a FORTH alapszótár első szavának névmezőcime található. Más szótárak egyszerűen a szótár nevének leirásával jelölhetők ki.

CURRENT
A CURRENT-ben annak a " szótárnév " szónak a csatolómezőcime van, amely által kijelölt szótárba a definiciók kerülnek. A CURRENT tartalma a " szótárnév DEFINITIONS " hatására változik. A CURRENT @ @ hatására igy a definiciókra kijelölt szotár első szavának névmezőcime kerül a veremre. Ezt a cimet egyszerűbben a **LATEST** szóval kaphatjuk meg. A LATEST definiciója: CURRENT @ @. Alapértelmezésben a változóban levő cimen a FORTH alapszótár első szavának névmezőcime található.

DP
A szótár soron következő szabad bájtjának cimét tartalmazza. Kiolvasására a HERE-t, megváltoztatására az ALLOT-ot szokás használni.

DPL
A beolvasott duplapontos egész számokban a tizedespont utáni számjegyek számát tartalmazza. Ha a számban tizedespont nem szerepelt, DPL értéke -1 lesz.

FENCE
Azt a szótárcimet tartalmazza, amely alatt a FORGET-tel nem lehet törölni. Tetszőlegesen módositható.

FLD
Eredetileg az output mezők méretével kapcsolatos változó. A legtöbb FORTH rendszer nem veszi figyelembe, igy szabadon felhasználható.

HLD
A beolvasott szöveg számmá alakitása esetén HLD tartalmazza a számkonvertáló által utoljára felhasznált karakter cimét.

IN (vagy egyes rendszerekben >IN)
Az aktuális szövegbuffer (ez vagy a terminál input buffer (TIB) vagy egy blokkpuffer) elejéhez képesti eltolást tartalmazza. A következő feldolgozandó karakterre mutat.

OUT
Az OUT-ban tárolt érték minden EMIT végrehajtásakor eggyel növekszik. Ily módon alkalmas a kivitt karakterek számolására. A változó értéke a FORTH inditásakor határozatlan, ezért használata előtt azt be kell állitani nullára. értéke bármikor lekérdezhető vagy módositható.

PREV
A változó tartalmazza a legutoljára használt diszkpuffer cimét.

R0
A visszatérési verem aljának cimét tartalmazza. Ha a visszatérési verem mutatója és az R0-ban levő cim megegyezik, akkor a verem üres. A visszatérési verem mutatójába irhatjuk az R0-ban tárolt cimet az **RP!** szó használatával. Ezzel a visszatérési verem törlését érhetjük el. A benne tárolt cim megegyezik UP-vel (l. a 6.4.3. pontot). A visszatérési verem az R0-tól kezdve a csökkenő, a felhasználói változók pedig a növekvő cimek felé helyezkednek el.

S0
Az adatverem aljának cimét tartalmazza. Az adatveremmutató aktuális értéke a veremre tehető az **SP@** szó segitségével. Igy a következő utasitásokkal: " S0 @ SP@ - " megkapjuk a verem aktuális mélységét bájtokban. Az S0-ban tárolt cimnek a veremmutatóba való irását az **SP!** szóval érhetjük el. Ez természetesen a verem törlését eredményezi.

Az R0 ill. S0 megváltoztatása a vermek áthelyezését jelenti. Egyes ritkán előforduló feladatoknál szükség lehet több verem fenntartására (például párhuzamos folyamatok esetén), de általában az R0 és S0 megváltoztatása nem ajánlatos.

SCR
Az éppen használt screen sorszámát tartalmazza.

STATE
A STATE tartalma mutatja, hogy a FORTH interpreter vagy compiler üzemmódban dolgozik-e. értéke végrehajtás közben nulla, fordítás alatt nemnulla.

TIB
A Terminál Input Buffer címét tartalmazza. Ez a cim megegyezik az S0-ban tárolttal, de mig a verem a kisebb, addig a buffer a nagyobb cimek felé növekszik.

USE
A következőnek használható (tehát a legrégebben beirt) blokkpuffer címét tartalmazza.

VOC-LINK
A legutoljára definiált szótár egyik mezőjének címét tartalmazza. A szótárnevek ezen keresztül fel vannak fűzve egy láncra.

WARNING
A szöveges hibaüzenetek háttértárról való beolvasását szabályozza (l. a 6.8. szakaszt: ERROR, WARNING).

WIDTH
A szavak nevének maximális hosszát tartalmazza (értéke általában 31).

6.7.2. Rendszerkonstansok

0 1 2 3
Mivel ezt a négy értéket igen gyakran használjuk, a FORTH ezeket konstansként definiálja. Szódefinicióban igy minden egyes előfordulásuk esetén két bájtot takaritunk meg, hiszen a szótárban a " LIT n " helyett (négy bájt) csak az értékül n-et adó konstans cime (két bájt) lesz.

#BUFF
A memóriapufferbeli blokkok számát adja meg.

B/BUF
A konstans egy diszkpuffer hosszát adja meg bájtokban.

B/SCR
Az egy screenben levő blokkok számát adja meg.

BL
értéke a szóköz ASCII kódjával, 32-vel egyezik meg.

C/L
A kiirásoknál az egy sorba irandó karakterek számát tartalmazza.

FIRST
Az első blokkpuffer cimét adja meg.

LIMIT
A diszkpuffer terület utáni első bájt címét adja meg.

6.8. Egyéb szavak

Ebben a fejezetben azon szavak rövid leirása szerepel, amelyeknek a programozás során csekély jelentőséggel birnak. A szavakat szinte kizárólag a FORTH rendszer használja, felhasználói programokban ritkán fordulnak elő. Mindemellett mégis érdemes néhány szót ejteni róluk, mert ismeretükben jobban átlátható a FORTH működése.

!CSP
A CSP változóba tölti a veremmutató aktuális értékét.

(.")
A ." szó forditja a szótárba a kiirandó szöveg elé. Futáskor a (.") az őt követő szöveget továbbitja az aktuális kiviteli eszközre. (A szövegre nézve a (.") körülbelül a LIT-nek felel meg).

(+LOOP)
A +LOOP forditja a szótárba a ciklusmag mögé. A (+LOOP) a ciklusváltozóhoz adja a vermen található növekményt, ellenőrzi a kilépési feltételt, és ha az nem teljesül, egy ugrást hajt végre a ciklusmag elejére.

(;CODE)
A ;CODE által forditott futási idejű eljárás, amely a definiálás alatt álló szó kódmezőjébe a ;CODE után álló gépi

kódú programrész cimét teszi.

(ABORT)
Alapértelmezésben egy ABORT-ot hajt végre. A felhasználó átirhatja a (ABORT) törzsét és igy egyéni hibakezelést valósithat meg.

(DO)
A DO helyezi a szótárba a ciklusmag elé. A (DO) a veremről a visszatérési veremre másolja a ciklust vezérlő paramétereket.

(LOOP)
A LOOP által a szótárba forditott futásidejű ciklusvezérlő szó. Eggyel növeli a ciklusváltozó értékét, ellenőrzi a kilépési feltételt. Ha az nem teljesül, a vezérlést a ciklusmag első szavára adja.

.CPU
Kiirja a processzor nevét, melyen az adott implementáció fut (pl. Z80).

+BUF cim1 --- cim2 offs
A cim1-ből kiszámitja a következő puffer cimét, és ezt cim2-ként a veremre helyezi. A verem tetején megadja cim2-nek a PREV-ben tárolt értéktől való távolságát (offs).

+ORIGIN n -- cim
A vermen egy egész számot vár. Ezt hozzáadja a FORTH rendszer kezdőciméhez, és az igy kapott cimet adja vissza a vermen.

?CSP
Hibajelzést ad, ha a veremmutató aktuális értéke nem egyezik meg a CSP-ben tárolt értékkel.

?ERROR f n --
Kiirja a verem tetején levő sorszámú hibaüzenetet, ha a verem második szintjén található érték (f) nullától különbözik (logikai igaz).

?LOADING
Hibajelzést ad, ha a FORTH rendszer végrehajtásakor nem a háttértárról olvassa be az adatokat.

?STACK
Hibaüzenetet ad, ha a verem túl- vagy alulcsordult.

?TERMINAL --- f
Megvizsgálja a billentyűzet állapotát. Ha végrehajtása előtt nyomtak le billentyűt, akkor igaz, ellenkezőleg hamis logikai értéket szolgáltat.

BYE
Általában ezzel az utasitással lehet kilépni a FORTH-ból, visszatérni az operációs rendszerbe.

ERROR n ---
Ha WARNING nemnegativ, végrehajt egy QUIT utasitást. Ezt megelőzően, ha WARNING nulla, akkor egy MSG # n üzenet jelenik meg. Ha WARNING pozitiv, akkor a 4. screen első sorától számitott n-edik sort irja ki hibaüzenetként. Ha n

nagyobb, mint az egy screenben levő sorok száma, akkor a screenek folytonos összefűzésével választja ki a szükséges sort. Ha WARNING értéke -1, csak egy (ABORT) utasitást hajt végre.

MESSAGE n ---
Hibaüzenetek kiirására használjuk. Ha a WARNING változóban nulla van, akkor **MSG # n** üzenetet kapunk, ahol n a hiba kódja. Ha WARNING nem nulla, akkor a 4. screen első sorától számitott n-edik sort irja ki hibaüzenetként. Ha n nagyobb, mint az egy screenben levő sorok száma, akkor a screenek folytonos összefűzésével választja ki a szükséges sort.

NOOP
Ez egy üres utasitás, definicója : NOOP ;

TASK
A felhasználó által definiált szótár kezdetét jelzi. Csak szimbólikus jelentése van.

F1. Feladatok megoldása

A 2. fejezet feladatainak megoldása:

2.10.1. SWAP ROT 4 ROLL . . . .

2.10.2. 2PICK = DUP + DUP 1+ PICK SWAP PICK

2.10.3. 2ROLL = DUP + DUP 1+ ROLL SWAP ROLL

2.10.4. PAD @ C@

2.10.5. KEY KEY MIN EMIT

```
2.10.6. PAD 1+ 10 EXPECT<R> ( szám beirása )
        BL PAD 11 + C!      ( szóköz elhelyezése )
        PAD NUMBER          ( szám felhozása a veremre )
        SWAP OVER DABS      ( konverzió előkészitése )
        <# # # # 44 HOLD # # # 44 HOLD # # # 44 HOLD
        #S SIGN #>
        TYPE                ( kiiratás )
```

A 3. fejezet feladatainak megoldása:

3.7.1. : PICK

```
( ... n_i+1 n_i n_i-1 ... i --- ... n_i+1 n_i n_i-1 ... n_i )
  DUP + SP@ + @ ;
```

3.7.2. : ROLL

```
( ... n_i+1 n_i n_i-1 ... i --- ... n_i+1 n_i-1 ... n_i )
SP@ S0 @ SWAP - 2 / ( = DEPTH )
>R          ( ... i )
```

```
DUP DUP     ( ... i i i )
R>          ( ... i i i depth )
<           ( ... i i i<depth )
>R          ( ... i i )
1           ( ... i i 1 )
>           ( ... i i>1 )
R>          ( ... i i>1 i<depth )
AND         ( ... i [i>1 és i<depth] )
IF 1+       ( ... i+1 )
   DUP      ( ... i+1 i+1 )
   PICK     ( ... i+1 n_i )
   SWAP     ( ... n_i i+1 )
   DUP +    ( ... n_i 2*[i+1] )
   SP@      ( ... n_i 2*[i+1] sp )
   + 2 -    ( ... n_i cim1=2*[i+1]+sp-2 )
   SP@ 1+   ( ... n_i cim1 cim2=sp+1 )
   SWAP     ( ... n_i cim2 cim1 )
   DO I @ I 2+ ! -2 +LOOP DROP
ELSE DROP
 ENDIF ;
```

3.7.3.

2.10.1. : FORDITVA SWAP ROT 4 ROLL . . . . ;

2.10.2. : 2PICK DUP + DUP 1+ PICK SWAP PICK ;

2.10.3. : 2ROLL DUP + DUP 1+ ROLL SWAP ROLL ;

2.10.4. : KIOLVAS PAD @ C@ ;

2.10.5. : MINKAR KEY KEY MIN EMIT ;

2.10.6. : BEOLV-KIIR
```
          PAD 1+ 10 EXPECT ( szám beolvasása )
          BL PAD 11 + C!   ( szóköz elhelyezése )
          PAD NUMBER       ( szám felhozása veremre )
```

```
    SWAP OVER DABS    ( konverzió előkészités )
    <# # # # 44 HOLD # # # 44 HOLD # # # 44 HOLD
    #S SIGN #>        ( konverzió )
    TYPE              ( kiiratás )
  ;
```

3.7.4. :
```
BEOLV-KONV ( --- d )
    PAD 1+ 80 EXPECT          ( RETURN-ig beolvas )
    PAD BEGIN 1+ DUP C@ 0=    ( PAD+1-től keres )
        UNTIL                 ( 0 karaktert )
    BL SWAP C!                ( 0 helyére szóköz )
    PAD NUMBER                ( számmá konvertál )
;
```

3.7.5. :
```
ATLAG ( n --- )
    0.                ( n d=0 )
    ROT               ( d=0 n )
    DUP               ( d=0 n n )
    >R                ( d=0 n )
    0 DO              ( d )
      BEOLV-KONV      ( d d1 )
      D+              ( d=d+d1 )
    LOOP
    R>                ( d n )
    DUP               ( d n n )
    ." a(z) " .       ( d n )
    M/ SWAP DROP      ( n1=d/n )
    S->D              ( d2 )
    ." darab szam atlaga:"
    12                ( d2 12 )
    D.R               ( kiirás )
;
```

3.7.6. :
```
HIBA ." Tulcsordulas" SP! QUIT ;
: HATV ( n1 n2 --- n3=n1^n2 )
  DUP 0=
```

```
IF 2DROP 1                      ( n2=0 -> n3=1 )
ELSE 1 SWAP 0                   ( n1 n3=1 n2 0 )
  DO                            ( n1 n3 )
    2DUP                        ( n1 n3 n1 n3 )
    *                           ( n1 n3 n4=n1*n3 )
    DUP                         ( n1 n3 n4 n4 )
    ROT                         ( n1 n4 n4 n3 )
    < IF HIBA ENDIF             ( n1 n3=n4 )
    LOOP
    SWAP                        ( n3 n1 )
    DROP                        ( n3 )
  ENDIF ;
```

3.7.7. :
```
TEKER                               ( n1 n2 --- n3=lko[n1,n2]
[ SMUDGE ]
SWAP                                ( n2 n1 )
OVER                                ( n2 n1 n2 )
MOD                                 ( n2 n3=n1 mod n2 )
?DUP
IF TEKER ENDIF                      ( ha n3>0 )
; SMUDGE
: LKO ( n1 n2 --- n3=lko[n1,n2] )
  ?DUP IF SWAP                      ( n2 n1 )
          ?DUP
          IF                        ( ha n1<>0 és n2<>0 )
             2DUP > IF
                      SWAP ( ha n2>n1 )
                    ENDIF
                    SWAP
                    TEKER
             ENDIF
          ENDIF
```

3.7.8. :
```
TEKER                               ( n1 n2 --- n3=lko[n1,n2]
SWAP                                ( n2 n1 )
```

```
  OVER                              ( n2 n1 n2 )
  MOD                               ( n2 n3=n1 mod n2 )
  ?DUP
  IF MYSELF ENDIF                    ( ha n3>0 )
;
: LKO ( n1 n2 --- n3=lko[n1,n2] )
  ?DUP IF SWAP                      ( n2 n1 )
          ?DUP
          IF                        ( ha n1<>0 és n2<>0 )
             2DUP > IF
                      SWAP ( ha n2>n1 )
                    ENDIF
                    SWAP
                    TEKER
          ENDIF
       ENDIF
;
```

3.7.9. Két megoldást adunk. Az első:

```
: MEG ( n1 --- )
( használata: n1 MEG n2 = n3 )
S->D          ( d1 )
13 TEXT       ( d1 ) ( CR-ig olvassa n2 karaktereit )
PAD           ( d1 PAD )
NUMBER        ( d1 d2 )
D+            ( d3=d1+d2 )
." = "
D. ;
```

A második megoldás a WORD szó használatára ad példát:

```
: MEG ( n1 --- )
( használata: n1 MEG n2 = n3 )
S->D          ( d1 )
13 WORD       ( d1 ) ( CR-ig olvassa n2 karaktereit )
```

```
HERE        ( d1 DP )
NUMBER      ( d1 d2 )
D+          ( d3=d1+d2 )
."  = "
D. ;
```

3.7.10. A megoldás egy több szóból álló program. A részletes megértést az Olvasóra bízzuk. Jelöléseink a következők: n az átalakitandó szám, dko a mindenkori dekádosztó (1000, 100, 10, 1), db a kiirandó egyesek darabszáma:

```
: 5.IR ( dko --- dko )
  ( dekádon belül 5-öst ir )
  DUP 1 = IF                    ." V"
          ELSE DUP 10 = IF      ." L"
                       ELSE ." D"
                       ENDIF
          ENDIF ;
: 1.IR ( dko --- dko )
  ( dekádon belül 1-est ir )
  DUP 1 = IF                                    ." I"
          ELSE DUP 10 = IF                      ." X"
                       ELSE DUP 100 = IF        ." C"
                                    ELSE ." M"
                                    ENDIF
                       ENDIF
          ENDIF ;
: 4.IR ( dko db --- dko )
  ( dekádon belül 4-est ir )
  DROP 1.IR 5.IR ;
: 9.IR ( dko db --- dko )
  ( dekádon belül 9-est ir )
  DROP 1.IR 10 * 1.IR 10 / ;
: EGYEB.IR ( dko db --- dko )
  ( dekádon belül 4-től és 9-től különböző számot
```

```
   ir )
   DUP 4 > IF >R 5.IR R> 5 - ENDIF
   ?DUP IF 0 DO 1.IR LOOP ENDIF ;
: DEKAD.IR ( dko db --- dko )
   ( dekádon belüli számot ir )
   ?DUP IF 4 = IF 4.IR
               ELSE DUP 9 = IF 9.IR
                           ELSE EGYEB.IR
                           ENDIF
               ENDIF
        ENDIF ;
: ROMAI ( n --- )
   ( a konverziót végrehajtó szó = főprogram )
   1000 4 0 DO SWAP OVER /MOD ROT SWAP
               DEKAD.IR 10 /
            LOOP 2DROP ;
```

F2. Példaprogram

Az alábbiakban egy hasznos FORTH programot ismertetünk, amely leforditott FORTH szavak visszafejtését végzi. Segitségével többek között elemezhetjük a FORTH alapszótár szavainak felépitését is. Hexadecimálisan kijelzi a memóriacimeket, és kiirja az ott levő szavak nevét. A gépi kódú szavakat külön jelzi. Használata a következő:

DIS szónév

Lássunk egy példát. Ennek során 3.3.2.5. pontban bemutatott, majd a 3.4. szakaszban elemzett VEGTELEN szó visszafejtését mutatjuk meg. Jól felismerhető a 3.9. ábrán látható struktúra.

```
: VEGTELEN BEGIN KEY EMIT AGAIN ; ok
DIS VEGTELEN


320B   :
320D   KEY
320F   EMIT
3211   BRANCH -> 320D
3215   ;S


ok
```

Természetesen a memóriacimek mindig mások lesznek. Most pedig lássuk a visszafejtő screenjeit:

```
SCR # 1
  0 ( FORTH visszafejtő )
  1 ( irta Ray Duncan )
  2
  3 DECIMAL : TASK ;
  4 ( CASE utasitás, l. a 4. fejezetet )
  5 ( irta Charles E. Eaker )
  6 : CASE ?COMP CSP @ !CSP 4 ; IMMEDIATE
  7 : OF 4 ?PAIRS COMPILE OVER COMPILE = COMPILE
  8       0BRANCH HERE 0 , COMPILE DROP 5 ; IMMEDIATE
  9 : ENDOF 5 ?PAIRS COMPILE BRANCH HERE 0 ,
 10          SWAP 2 [COMPILE] ENDIF 4 ; IMMEDIATE
 11 : ENDCASE 4 ?PAIRS COMPILE DROP
 12             BEGIN SP@ CSP @ = 0=
 13             WHILE 2 [COMPILE] ENDIF
 14             REPEAT CSP ! ; IMMEDIATE
 15 DECIMAL -->

SCR # 2
  0 ( FORTH visszafejtő, folytatás )
  1 ( speciális futásidejű szavak címének keresése )
  2 ' (LOOP)      2 - CONSTANT  LOOP.ADR
  3 ' LIT         2 - CONSTANT  LIT.ADR
  4 ' :         2 - @ CONSTANT  DOCOL.ADR
  5 ' 0BRANCH     2 - CONSTANT  0BRANCH.ADR
  6 ' BRANCH      2 - CONSTANT  BRANCH.ADR
  7 ' (+LOOP)     2 - CONSTANT  PLOOP.ADR
  8 ' (.")        2 - CONSTANT  PDOTQ.ADR
  9 ' C/L       2 - @ CONSTANT  CONST.ADR
 10 ' BASE      2 - @ CONSTANT  USERV.ADR
 11 ' USE       2 - @ CONSTANT  VAR.ADR
 12 ' (;CODE)    2 - CONSTANT  PSCODE.ADR
 13 0 VARIABLE QUIT.FLAG 0 VARIABLE WORD.PTR
 14 : N. DUP DECIMAL . SPACE HEX 0 ." ( " D. ." H )"
 15 DECIMAL ; -->
```

```
SCR # 3
  0 ( FORTH visszafejtő, folytatás )
  1 : PDOTQ.DSP ( füzér kiirása )
  2   WORD.PTR @ 2+ DUP >R DUP C@ + 1 - WORD.PTR !
  3   R> COUNT TYPE ;
  4 : WORD.DSP  ( adott cfa-jú szó nevének kiirása )
  5   3 - -1 TRAVERSE DUP 1+ C@ 59 = IF 1 QUIT.FLAG ! ENDIF
  6   DUP C@ 160 AND 128 = IF ID. ELSE 1 QUIT.FLAG ! DROP
  7   ENDIF ;
  8 : BRANCH.DSP ( ugrási eltolás kiirása ) ." -> "
  9   WORD.PTR @ 2+ DUP WORD.PTR ! @ 0 HEX D. DECIMAL ;
 10 : USERV.DSP ( felhasználói változó kiirása )
 11   ." Felhasznaloi valtozo, erteke = "
 12   WORD.PTR @ 2+ C@ 38 +ORIGIN @ + @ N. 1 QUIT.FLAG ! ;
 13 : VAR.DSP ( változó kiirása )
 14   ." Valtozo, erteke = "
 15   WORD.PTR @ 2+ @ N. 1 QUIT.FLAG ! ; -->

SCR # 4
  0 ( FORTH visszafejtő, folytatás )
  1 : CONST.DSP ( konstans kiirása )
  2   ." Konstans, erteke = "
  3   WORD.PTR @ 2+ @ 1 QUIT.FLAG ! ;
  4 : DIS ( főprogram )
  5   -FIND 0=
  6   IF 3 SPACES ." ? ismeretlen szo " CR
  7   ELSE DROP DUP DUP 2 - @ =
  8     IF 3 SPACES ." <gepi kod> " CR
  9     ELSE 0 QUIT.FLAG ! 2 - WORD.PTR ! CR CR
 10       BEGIN
 11         WORD.PTR @ DUP 0 HEX D. SPACE DECIMAL @
 12         CASE
 13          LIT.ADR      OF WORD.PTR @ 2+ DUP WORD.PTR !
 14                          @ N. ENDOF
 15          DOCOL.ADR    OF ." : " ENDOF -->
```

```
SCR # 5
  0 ( FORTH visszafejtő, folytatás )
  1             ØBRANCH.ADR OF ." ØBRANCH " BRANCH.DSP ENDOF
  2             BRANCH.ADR  OF ." BRANCH " BRANCH.DSP ENDOF
  3             LOOP.ADR    OF ." LOOP " BRANCH.DSP ENDOF
  4             PLOOP.ADR   OF ." +LOOP " BRANCH.DSP ENDOF
  5             PDOTQ.ADR   OF ." Szoveg kiiras: " PDOTQ.DSP
  6                            ENDOF
  7              USERV.ADR   OF USERV.DSP ENDOF
  8              VAR.ADR     OF VAR.DSP ENDOF
  9              CONST.ADR   OF CONST.DSP
 10              PSCODE.ADR  OF WORD.PTR @ @ WORD.DSP
 11                             1 QUIT.FLAG ! ENDOF
 12            DUP WORD.DSP
 13            ENDCASE CR
 14            2 WORD.PTR +! QUIT.FLAG @ ?TERMINAL OR
 15          UNTIL -->

SCR # 6
  0 ( FORTH visszafejtő, folytatás )
  1       ENDIF
  2     ENDIF CR ;
  3     CR ." DIS szonev " CR
  4 ;S
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

F3.1. 8080 CPM FORTH 2.2

A 8080 CPM FORTH 2.2 a fig-FORTH-szal kompatibilis rendszer. A virtuális memóriakezelés során szabványos CP/M fájlokat használ, igy a screenek a rendszerlemezen is tárolhatók. Az alábbiakban ismertetjük a fig-FORTH-tól eltérő lemezkezelő szavait, majd a szövegszerkesztő utasitásait.

DELETE cim --- j

A megadott FCB cimű fájl törlését végzi. Ha sikerült, jelző=0.

FCBIN --- cim

Bemeneti (input) fájl számára FCB (File Control Block) cimet tesz a veremre. A megadott mező első 33 helye szabványos CP/M FCB, további három bájt véletlen rekord cimmező, majd két bájton a fájl hossza található. Ezt követően 128 bájtos puffer van lefoglalva egy szektornyi információ számára.

FCBOUT --- cim

Kimeneti (output) fájl számára FCB cimet tesz a veremre. A lefoglalt mező felépitése megegyezik az FCBIN-nél leirtakkal.

FDOS n1 n2 --- n3 n4

A BDOS funkciók elérését teszi lehetővé az 5-ös cimen található belépési pont meghivásával. A C regiszterbe kerül n1 alsó 8 bitje (funkciókód), a DE-be pedig n2. A visszatérés utáni HL tartalom n3-ba, DE pedig n4-be kerül.

FILEIN fájlnév

A virtuális memóriakezeléshez megnyitja a FILEIN-t követő nevű fájlt beolvasásra (pl.: FILEIN ASSEMBLER.SCR).

FILENAME fájlnév cim ---

Az FCBIN és FCBOUT utasitásokhoz használható, létrehozza a FILENAME-t követő nevű fájl File Control Block-ját (pl.: FCBIN FILENAME NEV.SCR)

FILEOUT fájlnév

A virtuális memóriakezeléshez megnyitja a FILEOUT-ot követő nevű fájlt irásra (pl.: FILEOUT LO.SCR).

I=O

Az irásra megnyitott fájlt irásra is megnyitja.

NEWFILE fájlnév n ---

Létrehoz egy n üres screent tartalmazó fájlt, és ezt irásra meg is nyitja (pl.: 10 NEWFILE PROBA.SCR).

O=I

Az olvasásra megnyitott fájlt irásra is megnyitja.

OPEN cim ---

Virtuális memóriakezeléshez megnyitja a cimen található FCB-jű fájlt.

PARSE-FILENAME cim1 cim2 ---
A cim2 cimen levő FCB a cim1-re másolódik.

READ cim1 --- cim2
A cim1-en található FCB-jű fájl beállitott szektorát a mező utolsó 128 bájtjos pufferébe másolja. Sikeres átvitel esetén cim2 a puffer kezdőcime, egyébként 0.

SAVE-INIT
A felhasználói szótárat beolvasztja az alapszótárba és kiirja, hogy igy hány lemezblokkot foglal el a FORTH a lemezen. Ezután BYE paranccsal kilépve CP/M-be, kimenthetjük a megadott blokkszámú programot.

WRITE cim1 --- cim2
A cim1-en található FCB-jű fájl beállitott szektorába beirja a mező utolsó 128 bájtjos pufferének tartalmát. Sikeres átvitel esetén cim2 a puffer kezdőcime, egyébként 0.

Most ismertetjük az **EDITOR** utasitásait:

L
Az aktuális screent listázza (List).

N
A sorrendben következő screent olvassa be (Next).

B

A sorrendben előző screent hozza be a memóriába (Back).

U szöveg

A szöveget az inzert pufferbe, és ezzel egyidejűleg a kurzor sorát követő sorba másolja. Az alatta levő sorok eggyel lejjebb csúsznak, a 15-ös sor elvész. Szöveg nélkül kiadva az inzert puffer utolsó tartalmával végzi el a műveletet (Under).

T n ---

Az n-edik sort a képernyőre listázza (Type).

P szöveg

A P-t követő szöveget az inzert pufferbe, és ezzel egyidejűleg az aktuális sorba másolja. A sor eredeti tartalma elvész (Put).

X

Az aktuális sor tartalmát az inzert pufferba teszi, és az összes alatta levőt eggyel feljebb lépteti. A 15-ös sor üres lesz (eXtract).

F szöveg

Az F-et követő szöveget a keresőpufferbe másolja, majd annak első előfordulását keresi a kurzor mögött a screenben. Sikeres keresés esetén a szöveg mögé állitja a kurzort, egyébként hibajelzést ad. Szöveg nélkül kiadva az utoljára keresett szöveget keresi. (Find).

E

A kurzortól visszafelé annyi karaktert töröl, ahány karakterből a keresőpufferben levő szöveg áll (Erase)..

D szöveg

A D-t követő szöveget a keresőpufferbe másolja, megkeresi a kurzor utáni első előfordulását, majd sikeres keresés esetén törli a szövegből (Delete).

TILL szöveg

A TILL-t követő szöveget a keresőpufferbe másolja, megkeresi a kurzor utáni első előfordulását, majd sikeres keresés esetén a kurzortól a szöveg végéig törli a screen tartalmát.

I szöveg

Az I-t követő szöveget az inzert pufferbe, majd a kurzor mögé másolja. A sornak a kurzor mögötti része jobbra tolódik, a 64. pozición túliak elvesznek (Insert).

R szöveg

Az R-et követő szöveget az inzert pufferbe másolja, majd a kurzortól visszafelé törli a keresőpufferben tárolt szöveg hosszának megfelelő számú karaktert, és helyére másolja az inzert puffer tartalmát (Replace).

M n1 n2

Az aktuális sort az inzert pufferbe, majd az n1. screen n2. sora mögé másolja egy U utasitással. Az aktuális screen nem változik, a kurzor egy sorral lejjebb kerül (Move).

S szöveg n ---

Az S-et követő szöveget a keresőpufferbe másolja, majd az aktuális screen első karakterétől kezdődően az n-1. screen zárókarakteréig keresi annak előfordulásait. Minden előfordulás esetén kiirja az adott screen számát és az adott sort (Search).

F3.2. Commodore C-64 FORTH

A C-64 FORTH a fig-FORTH-szal kompatibilis rendszer. A következőkben néhány plusz szolgáltatását, valamint a lemezegységet kezelő és szövegszerkesztő szavait ismertetjük. Meg kell említenünk, hogy a virtuális memóriakezelés során a lemezt nem fájlstruktúrában használja, igy erre a célra külön diszket kell formatálni.

(DERROR)

Diszkhiba esetén hajtódik végre.

-BCD n1 --- n2

n1-et BCD kódra konvertálja. 0<=n<=99.

-DISC cim s p l r/w --- f

Blokkok átvitelét vezérli a memória és a lemezegység között. Az átvitel a cimen kezdődő blokk és az l lemezegység p pályájának s szektora között történik. r/w=0 irást, r/w=1 olvasást jelöl. Hibátlan átvitel esetén f hamis lesz, ellenkező esetben igaz.

?DISC

Beolvassa a lemezegység státusz/hiba kódját és tárolja a 08BFH cimen. Az üzenet hossza 08C0H-n található.

?HOME

Ha a kurzor a jobb alsó sarokban van, akkor azt a bal felsőbe viszi.

?TERMINAL --- f

f igaz, ha a STOP billentyű le van nyomva.

BOOT-UP

A virtuális memóriakezelő rendszer kezdőállapotra hozása. Inicializálja a lemezegységet, törli a pufferekat és betölti az első screent.

CKEY --- n

Hasonló a KEY utasitáshoz, de a karakter bevitelére villogó kurzorral várakozik.

CLOAD --- f

Beolvas a kazettás egységről egy szótárt. Ha a beolvasás sikeres volt, f hamis lesz, egyébként pedig a hibakód értékét veszi fel.

CLOSE n ---

Lezárja az n sorszámú fájlt.

CSAVE --- f

A kazettára irja a felhasználói szótárt és a rendszerváltozók értékét. Ha a kivitel sikeres volt, f hamis lesz, egyébként pedig a hibakód értékét veszi fel. A fájl neve a NAME szóval adható meg.

DISC

Inicializálja a mágneslemez kezelését, új lemez behelyezésekor kell kiadni.

HOME

A kurzort a bal felső sarokba viszi.

HPIN n --- f

Az n sorszámú fájlt kijelöli kiviteli eszközként. Ha sikerül, f hamis lesz, ellenkező esetben a hibakód értékét veszi fel.

HPOFF

Beviteli eszközként a billentyűzetet, kiviteli eszközként a képernyőt jelöli ki.

HPOUT n --- f

Az n sorszámú fájlt kijelöli beviteli eszközként. Ha sikerül, f hamis lesz, ellenkező esetben a hibakód értékét veszi fel.

NAME cim n ---

n hosszúságú fájlnevet olvas be és helyez el cimtől kezdve a későbbi IEEE fájlműveletek számára. Ha fájlnév nem szükséges, akkor cimnek és n-nek nullának kell lennie.

OPEN n1 n2 n3 --- f

Az n1 sorszámú fájlt nyitja meg az n2 számú logikai egységen n3 másodlagos cimmel. Előtte a fájl nevét a NAME utasitással be kell állitani. Sikeres megnyitás esetén f hamis, ellenkezőleg a hibakód értéke.

S>P cim n ---

A cimtől elhelyezkedő n darab karaktert képernyőkódról programkódra konvertálja.

STATUS --- n

Az IEEE fájl állapotkódját adja meg. n alsó bájtja bitjeinek jelentése:

0. bit - időzitési hiba irásnál.
1. bit - időzitési hiba olvasásnál.
2. bit - rövid blokk.
3. bit - hosszú blokk.
4. bit - javithatatlan olvasási hiba.
5. bit - kontroll-összeg hiba.
6. bit - fájl vége.
7. bit - készülék hiányzik.

A továbbiakban a C-64 FORTH **EDITOR** szótárának szavait ismertetjük.

-MOVE cim n ---

A cimtől kezdődő szöveget az aktuális screen n-edik sorába viszi.

>BLKS

A képernyő tartalmát az aktuális screenbe viszi.

>SCR

Kiirja az aktuális screen tartalmát.

ATA@ --- cim

Megadja a kurzor pillanatnyi poziciójának cimét.

B

Áttér az aktuálisnál eggyel kisebb sorszámú screenre (Back).

CLEAR n ---

Az n-edik screent szóköz karakterekkel tölti fel.

D n ---

Az aktuális screenből törli az n-edik sort. Az n-edik utáni sorok feljebb tolódnak és az utolsó sor szóközökkel töltődik fel (Delete).

FREE n1 n2 ---

Kiirja az n1 és n2 sorszámok közé eső szabad screenek sorszámait. A screen szabad, ha első bájtja nulla.

H n ---

Az aktuális screen n-edik sorát PAD-től kezdve leteszi. A sort szóközökkel 40 karakter hosszúra egészíti ki (Hold).

I n ---

A PAD-től kezdődő szöveget az aktuális screen n-edik sorába irja. A régi n-edik, valamint az azt követő sorok lejjebb tolódnak. Az utolsó sor elveszik (Insert).

L

Sorszámokkal együtt kilistázza az aktuális screent (List).

LINE n --- cim

Megadja az aktuális screen n-edik sorának cimét.

N

SCR tartalmát eggyel növeli, igy ezután a következő screen lesz az aktuális screen (Next).

P szöveg n ---

Az utasitás a szöveg-et az aktuális screen n-edik sorába és a PAD-től kezdve elhelyezi (Put).

PAPER

Megnyitja a nyomtatót 4-es sorszámú fájlként, és kijelöli aktuális kiviteli eszköznek.

R n ---

Az aktuális screen n-edik sorát a PAD-től kezdődő szöveggel helyettesiti (Replace).

S n ---

Az aktuális screenbe beszúr egy üres sort, amelynek sorszáma n lesz. Az n-ediket követő sorok lejjebb tolódnak, és az utolsó sor elvész (Spread).

SCREENS n1 n2 n3 ---
n3 darab screent másol át n1-től kezdve n2-be (n1->n2, n1+1->n2+1 stb.).

SHOW n1 n2 ---
Kiirja az n1 és n2 közé eső összes TRIAD-ot.

TCOLOR
Rendszerváltozó, amely a szerkesztéskor érvényes betűszin kódját tartalmazza. Tetszőlegesen módositható.

VIDEO
Lezárja a 4-es sorszámú nyomtatófájlt, és kiviteli eszköznek a képernyőt jelöli ki.

X
Hatására a képernyő-orientált screenszerkesztőbe léphetünk be. A szerkesztés az aktuális screenen történik. A szerkesztés során a Commodore eredeti szövegszerkesztőjében megszokott billentyűket használhatjuk (kurzormozgatók, INST, DEL stb.). A STOP billentyű lenyomására a módositott screen a diszkpufferbe iródik, és ezután kilépünk a szerkesztőből. A SHIFT/STOP billentyű lenyomása szintén kilépést eredményez, de a screen tartalma nem iródik vissza a pufferbe.

ZERO n ---
Az n-edik sorszámú screent ASCII nulla kódú karakterekkel tölti fel. Ezzel az adott screen szabaddá válik.

F3.3. Spectrum ABERSOFT FORTH

A 48k ZX SPECTRUM gépekre fejlesztette ki az Abersoft cég ezt a fig-FORTH rendszert. Virtuális memóriakezelésre a microdrive-ot használja a fig-FORTH-nak megfelelően. Mivel nem fájlstruktúrában dolgozik, az erre szánt cartridge másra nem használható. Microdrive nélküli rendszerben a memóriában egy úgynevezett RAM-DISC szolgál 10 darab screen tárolására. Ennek tartalmát külön utasitásokkal lehet szalagra menteni, illetve onnan betölteni. Speciális utasitásai a következők:

AT n1 n1 ---
A kurzort az n1-edik sor n2-ik oszlopába viszi.

ATTR n1 n2 --- b
A verem tetején a képernyő n1-edik sorának n2-ik oszlopába tartozó állapotbájtot adja meg.

BLEEP n1 n2 ---
n1 hosszúságú, n2 magasságú hangot generál a hangszóróban. Ezek a paraméterek nem azonosak a BASIC BEEP utasitásával!

BRIGHT n ---
Azonos a BASIC BRIGHT utasitással.

BORDER n ---
Azonos a BASIC BORDER utasitással.

CASE

A 4.3. szakaszban leirt FORTH struktúrához tartozik.

CLS

Azonos a BASIC CLS utasitással.

DRAW n1 n2 ---

Azonos a BASIC DRAW utasitással.

ENDCASE

A 4.3. szakaszban leirt FORTH struktúrához tartozik.

ENDOF

A 4.3. szakaszban leirt FORTH struktúrához tartozik.

FLASH n ---

Azonos a BASIC FLASH utasitással.

FREE --- n

A szabad memóriaterület méretét teszi a veremre.

GOVER n ---

Azonos a BASIC OVER utasitással.

INIT-DISC

A RAM-DISC-et feltölti szóköz karakterekkel, és megnyitja a virtuális memóriakezelésre.

INK n ---
Azonos a BASIC INK utasitással.

INKEY --- n
A lenyomott billentyű kódját helyezi a veremre. Ha nincs lenyomva billentyű, n=255.

INP n1 --- n2
Az n1-edik portról belvasott n2 értéket helyezi a veremre.

INVERSE n ---
Azonos a BASIC INVERSE utasitással.

LINK j ---
A printert be- (j=1) illetve ki- (j=0) kapcsolja.

LOADT
Szalagról beolvassa a RAM-DISC területre a korábban SAVET-tel elmentett információt.

MON
Visszatérés BASIC-be.

OF
A 4.3. szakaszban leirt FORTH struktúrához tartozik.

OUTP n1 n2 ---
Az n2-ik portra kiküldi n1 alsó 8 bitjét.

PAPER n ---
Azonos a BASIC PAPER utasitással

PLOT n1 n2 ---
Azonos a BASIC PLOT utasitással.

POINT n1 n2 --- j
Azonos a BASIC POINT utasitással.

SAVET
DISC néven kimenti a RAM-DISC tartalmát szalagra.

SCREEN n1 n2 --- n
Azonos a BASIC SCREEN$ utasitással.

SIZE --- n
A veremre helyezi a FORTH rendszer szótára által elfoglalt memória méretét bájtokban.

UDG --- cim
A felhasználó által definiálható karakterek kezdőcimét adja meg.

VERIFY
A szalagon tárolt információ ellenőrzésére szolgál.

Az alábbi szavak a FORTH-79 megfelelői:

2CONSTANT
2DROP
2OVER
2ROT
2SWAP
2VARIABLE
U.R

Most ismertetjük az ABERSOFT FORTH-hoz használható **EDITOR** utasitásait, majd az editor screeneket.

CLEAR n ---
Az n-edik screent feltölti szóköz karakterekkel, és kijelöli a szerkesztéshez.

L
Listázza az aktuális screent (List).

D n ---
Az aktuális screen n-edik sorát törli, és a PAD-en tárolja (Delete).

E n ---
Az aktuális screen n-edik sorát feltölti szóköz karakterekkel (Erase).

H n ---
Az n-edik sort PAD-re másolja (Hold).

I n ---
A PAD-en tárolt szöveget az n-edik sorba másolja, az alatta levőket eggyel lejjeb csúsztatja, igy a 15. sor elvész (Insert).

P szöveg n ---
A P-t követő szöveget a screen n-edik sorába másolja (pl.: 3 P : LO 1 . ; a LO definicóját az aktuális screen 3. sorába viszi) (Put).

R n ---
Az n-edik sor tartalmát felülirja a PAD-en tárolt szöveggel (Replace).

S n ---
Az n-edik sor elé egy üres sort szúr be. Az alatta levőket eggyel lejjeb csúsztatja, igy a 15. sor elvész (Spread).

T n ---
Az n-edik sort kiirja, és PAD-en tárolja (Type).

M n ---
A kurzort az adott sorban n karakterhellyel elmozgatja, és kiirja a sort. A kurzort külön jelöli (Move).

TOP

A kurzort az aktuális screen 0. sorának elejére állitja.

B

A kurzort a PAD-en levő szöveg hosszának megfelelően visszalépteti (Back).

C szöveg

A C-t követő szöveget a kurzor sorába a kurzor mögé másolja (Copy).

F szöveg

Az F-et követő szöveget PAD-re másolja, és az aktuális screenben keresi a kurzor mögött (Find).

N

A PAD-nél levő szöveg kurzor utáni első előfordulását keresi az aktuális screenben (Next).

TEXT szöveg n ---

A TEXT-et követő szöveget az n-nek megfelelő lezáró karakterig a PAD-re másolja.

TILL

A kurzor sorában a TILL-t követő szövegig törli a sor tartalmát.

X szöveg
A kurzor sorában törli az X-et követő szöveg első előfordulását (eXtract).

```
SCR # 1
  0 ( SPECTRUM FORTH EDITOR #1 ) FORTH DEFINITIONS HEX
  1 : TEXT HERE C/L 1+ BLANKS WORD HERE PAD C/L 1+ CMOVE ;
  2 : LINE DUP FFF0 AND 17 ?ERROR SCR @ (LINE) DROP ;
  3 VOCABULARY EDITOR IMMEDIATE HEX
  4 WHERE DUP B/SCR / DUP SCR ! ." SCR# " DECIMAL . SWAP
  5 C/L /MOD C/L * ROT BLOCK + CR C/L TYPE CR HERE C@ -
  6 SPACES 5E EMIT [COMPILE] EDITOR QUIT ;
  7 EDITOR DEFINITIONS
  8 : #LOCATE R# @ C/L /MOD ; : #LEAD #LOCATE LINE SWAP ;
  9 : #LAG #LEAD DUP >R + C/L R> - ; : -MOVE LINE C/L
 10 CMOVE ; : H LINE PAD 1+ C/L DUP PAD C! CMOVE ;
 11 : E LINE C/L BLANKS ;
 12 : S DUP 1 - 0E DO I LINE I 1+ -MOVE -1 +LOOP E ;
 13 : D DUP H 0F DUP ROT DO I 1+ LINE I -MOVE LOOP E ;
 14
 15 -->
```

```
SCR # 2
  0 ( SPECTRUM FORTH EDITOR #2 )
  1 : M R# +! CR SPACE #LEAD TYPE 5F EMIT #LAG TYPE #LOCATE
  2 . DROP ;
  3 : T DUP C/L * R# ! DUP H 0 M ; : L SCR @ LIST 0 M ;
  4 : R PAD 1+ SWAP -MOVE ; : P 1 TEXT R ;
  5 : I DUP S R ; : TOP 0 R# ! ;
  6 : CLEAR DUP FIRST ! SCR ! 100 DO FORTH I EDITOR E
  7 LOOP ; FORTH DEFINITIONS : 2DROP DROP DROP .;
  8 : 2SWAP ROT >R ROT R> ; EDITOR DEFINITIONS
  9 : -TEXT SWAP -DUP IF OVER + SWAP DO DUP C@ FORTH I
 10 C@ - IF 0= LEAVE ELSE 1+ THEN LOOP ELSE DROP 0= THEN ;
 11 : MATCH >R >R 2DUP R> R> 2SWAP OVER + SWAP DO 2DUP
 12 FORTH I -TEXT IF >R 2DROP R> - I SWAP - 0 SWAP 0 0
 13 LEAVE THEN LOOP 2DROP SWAP 0= SWAP ;
 14 : 1LINE #LAG PAD COUNT MATCH R# +! ;
 15 -->

SCR # 3
  0 ( SPECTRUM FORTH EDITOR #3 )
  1 : FIND BEGIN 3FF R# @ < IF TOP PAD HERE C/L 1+ CMOVE
  2 0 ERROR ENDIF 1LINE UNTIL ;
  3 : DELETE >R #LAG + FORTH R - #LAG R - R# +! #LEAD
  4 + SWAP CMOVE R> BLANKS ;
  5 : N FIND 0 M ; : F 1 TEXT N ;
  6 : B PAD C@ MINUS M ; : X 1 TEXT FIND PAD C@ DELETE
  7 0 M ;
  8 : TILL #LEAD + 1 TEXT 1LINE 0= 0 ?ERROR #LEAD + SWAP
  9 - DELETE 0 M ;
 10 : C 1 TEXT PAD COUNT #LAG ROT OVER MIN >R FORTH R R#
 11 +! R - >R DUP HERE R CMOVE HERE #LEAD + R> CMOVE
 12 R> CMOVE 0 M
 13
 14 FORTH DEFINITIONS DECIMAL
 15 ;S
```

F3.4. ZX SPECTRUM 48k Floating Point FORTH

A FORTH rendszer a LOAD "" utasitással tölthető be kazettáról. A lebegőpontos FORTH nem " ok " üzenetet ir ki, hanem a következő sorba egy ...> promptot. Interpreter módban is kell a parancssorok végére a pontosvessző.

```
...>6 2 + . ; 8
...>
```

Minden szám 6 bájtot foglal el a vermen. Lebegőpontos számoknál 5 bájt értékes, 1 bájt értéktelen, egész számoknél 2 bájt értékes, 3 bájt nulla, 1 bájt értéktelen. A lebegőpontos számokban vagy tizedespont (14.9, -.Ø73) vagy E van (2E9, 7E-3, -8.4E7). Az egész számok csak számjegyeket és előjelet tartalmaznak (1, 973, -1961). Lebegőpontos szám helyett minden esetben állhat egész, mert erre a konverzió automatikus. Egész helyett lebegő viszont soha nem alkalmazható. A lebegőpontos számokat 1 betű jelzi, a többi jelölés a megszokott lesz. Az ENTER billentyű lenyomását <enter>-rel jelezzük.

A szavak neve tetszőlegesen hosszú lehet, de csak az első hat karakter kerül tárolásra. A kis- és nagybetük különbözőnek számitanak. A lebegőpontos FORTH a SPECTRUM eredeti hibajelzéseit használja. Hiba esetén GOTO 4050 vagy GOTO 3 utasitással térhetünk vissza a FORTH-ba.

A következőkben közöljük a lebegőpontos FORTH szavait, témakörök szerint csoportositva.

z

z<enter> hatására a képernyő tartalma a nyomtatóra kerül.

pron

pron<enter> hatására minden további szöveg a nyomtatóra kerül és a képernyőn nem jelenik meg.

proff

proff<enter> semlegesiti pron hatását.

s

Az egész FORTH rendszer és a felhasználói szótár szalagra mentését az s<enter> utasitással kezdeményezhetjük. Ezután meg kell adni a program nevét, ami legfeljebb 10 karakterből állhat, és nem tartalmazhat szóközt. A mentés három részletben történik. Ezután vissza kell tekerni a program elejére, és egy ellenőrző olvasás történik. Ha nincs hiba, "OK" üzenet jelenik meg, ha van, akkor break GOTO 2010 után újból meg kell kisérelni a mentést.

. n ---

Lebegőpontos szám kiirása.

%. n ---

Egész szám kiirása.

FIELD n ---

A következő 16 karakteres mezőre lép, és kiirja a számot jobbra tömöritve.

%FIELD n ---

A következő 8 karakteres mezőre lép, és kiirja a számot jobbra tömöritve.

INT 1 --- n
Lebegőpontos szám egészrészét képzi

SQR 11 --- 12
12 11 négyzetgyöke.

RND --- 1
Véletlen számot generál. 0<=1<1.

SINR 11 --- 12
11 szinuszát számolja. 11 radiánban értendő.

COSR 11 --- 12
11 koszinuszát számolja. 11 radiánban értendő.

TANR 11 --- 12
11 tangensét számolja. 11 radiánban értendő.

ASNR ACSR ATNR 11 --- 12
A megfelelő szögfüggvények inverz függvényei. Az eredmény (12) radiánban képződik.

SIND COSD TAND 11 --- 12
Szögfüggvények, ahol 11 fokokban adandó meg.

ASND ACSD ATND 11 --- 12
Inverz szögfüggvények, ahol az eredmény (12) fokokban képződik.

LN 11 --- 12
11 természetes (e alapú) logaritmusa. 11>0.

EXP 11 --- 12
A logaritmusfüggvény inverze. 12=e^11.

^ 11 12 --- 13
Hatványozás. 13=11^12.

+ - * / 1+ 1- 2+ 2- NEGATE ABS
Funkciójukban a FORTH-79 hasonló szavaival egyeznek meg, de bemenő és kimenő paramétereik lebegőpontos számok.

< > = <= >= <> 11 12 --- f
Logikai műveletek két lebegőpontos szám összehasonlítására. f=0 a hamis, f=1 az igaz érték.

0< 0> 0= 1 --- f
Lebegőpontos szám és a nulla összehasonlítása.

MIN 11 12 --- 13
13=min(11,12).

MAX 11 12 --- 13
13=max(11,12).

AND 11 12 --- 13
Ha 12 nulla, akkor 13=0, egyébként 13=11.

OR 11 12 --- 13
Ha 12 nemnulla, akkor 13=1, egyébként 13=11.

NOT 11 --- 12
Ha 11 nulla, akkor 12=1, egyébként 12=0.

%+ %- %* %/ %1+ %1- %2+ %2- %NEGATE %ABS
%< %> %= %<= %>= %<>
%0< %0> %0=
%MIN %MAX
%AND %OR %NOT
A lebegőpontos műveletek egész számokra vonatkozó megfelelői.

DROP DUP ?DUP SWAP OVER PICK ROT ROLL
A FORTH-79 megfelelő veremműveletei, csak itt egy logikai veremszint nem kettő, hanem hat bájt.

BEGIN...UNTIL
BEGIN...WHILE...REPEAT
Megegyeznek a FORTH-79 hasonló ciklusaival.

DO...LOOP
A kezdő- és a végérték egyaránt lehet lebegőpontos szám.

DO...+LOOP

A kezdő-, vég- és lépésközérték lebegőpontos, de a lépésköz csak pozitiv lehet.

DO...-LOOP

Ugyanaz, mint a DO...+LOOP, de itt a lépésköz (amely szintén csak pozitiv lehet) levonódik a ciklusváltozóból.

I --- l

A lebegőpontos ciklusváltozót teszi a veremre.

J --- l

Egymásba ágyazott ciklusoknál belülről a második ciklus ciklusváltozóját adja meg.

K --- l

Belülről a harmadik ciklus változóját adja meg.

LEAVE

A lebegőpontos ciklusváltozót a végértékre állitja.

EXITLP

Végrehajtása a DO ciklusból való azonnali kilépést eredményez.

%DO %LOOP %+LOOP %I %LEAVE EXIT%L

A lebegőpontos ciklusok megfelelői egész kezdő-, vég- és lépésközértékekkel. A J, K és -LOOP szavaknak nincs egész megfelelője.

IF...ELSE...THEN
Ugyanaz, mint a FORTH-79-ben, de itt az ENDIF nem használható, csak a THEN.

BEEP 11 12 ---
11 időtartamú, 12 hangmagasságú hangjelzést generál. Megegyezik a hasonló BASIC utasitással.

BLEEP n1 n2 ---
n1 időtartamú, n2 hangmagasságú hangot állit elő. A két egész paraméter hatása nem egyezik meg a BEEP hasonló lebegőpontos paramétereivel, ezért megfelelő hangot csak kisérletezés útján kaphatunk.

CLS
Törli a képernyőt, a kurzort a bal felső sarokba állitja.

AT 11 12 ---
Az 11. sor 12. oszlopába viszi a kurzort.

TAB 1 ---
A kurrens sor 1-edik oszlopába viszi a kurzort.

INK PAPER BORDER BRIGHT FLASH INVERSE
Megegyeznek a hasonló BASIC utasitásokkal, csak ezek a paraméterüket a vermen várják.

ATTR n1 n2 --- n3
n3 az n1 sor n2 oszlopának szinattributuma lesz.

PLOT 11 12 ---
Berajzolja az (11,12) koordinátájú pontot.

CIRCLE 11 12 13 ---
Egy (11,12) középpontú, 13 sugarú kört rajzol.

DRAW 11 12 ---
Vonalat húz az utolsó PLOT vagy DRAW utasitás által meghatározott (x,y) és az (x+11,y+12) koordinátájú pontok között.

POINT n1 n2 --- f
f igaz, ha az (n1,n2) koordinátájú pont be van kapcsolva, ellenkező esetben f hamis.

A B C D ... G H L M ... X Y Z
Változókat a lebegőpontos FORTH-ban nem definiálhatunk. Helyettük a fenti 23, előre definiált változó használható a megszokott módon (az I, J, K kivételével). Mind lebegőpontos, mind egész számok tárolására alkalmasak.

Ebben a FORTH-ban egyébiránt konstansdefiníció is csak igy lehetséges:

```
: PI 3.141592654 ;
```

INKEY --- n
Az éppen lenyomott billentyű kódját teszi a veremre. Ha nincs lenyomva billentyű, n=255-öt ad vissza.

GET --- n
A FORTH-79 KEY utasitásával egyezik meg.

@ n --- l
Az n cimen levő lebegőpontos számot teszi a veremre.

%@ n1 --- n2
Az n1 cimen levő egész számot teszi a veremre.

C@ n --- b
Az n cimen levő bájtot teszi a veremre. (1 bájt értékes, 4 bájt nulla, 1 bájt értéktelen).

P@ n --- b
Az n portcimről olvas be egy bájtot.

! l n ---
Az l lebegőpontos számot irja az n cimre.

%! n1 n2 ---
Az n1 egész számot irja az n2 cimre.

C! b n ---
A b bájtot irja az n cimre.

P! b n ---
A b bájtot irja az n portcimre.

? n ---
Kiirja az n cimen található lebegőpontos számot.

%? n ---
Kiirja az n cimen található egész számot.

C? n ---
Kiirja az n cimen található bájtot.

FILL n1 n2 b ---
Betölt n2 darab b bájtot az n1 cimtől kezdve.

ERASE n1 n2 ---
n2 darab nullabájtot tölt n1 cimtől kezdve.

DELETE n1 n2 ---
c1 cimtől kezdve n2 darab bájtot a szóköz ASCII kódjával (32) tölt fel.

MOVE n1 n2 n3 ---
n3 darab lebegőpontos számot (számonként öt bájtot) tölt át n2 cimre n1 cimtől kezdve.

%MOVE n1 n2 n3 ---
n3 darab egész számot (számonként két bájtot) tölt át n2 cimre n1 cimtől kezdve.

CMOVE n1 n2 n3 ---
n3 darab bájtot tölt át n2 cimre n1 cimtől kezdve.

DUMP n1 n2 ---
Kinyomtatja n1 darab lebegőpontos szám értékét n2 cimtől kezdve.

%DUMP n1 n2 ---
Kinyomtatja n1 darab egész szám értékét n2 cimtől kezdve.

CDUMP n1 n2 ---
Kinyomtatja n1 darab bájt értékét n2 cimtől kezdve.

-TRAIL n1 n2 --- n1 n3
Megegyezik a FORTH-79 -TRAILING utasitásával.

COUNT n --- n+1 b
Egy bájtot olvas be az n cimről, és a veremre teszi. A cimet eggyel megnöveli.

QUERY
Karaktereket olvas be és tárolja őket PAD-től kezdődően. A beolvasást befejezi, ha megnyomták az <enter>-t, vagy már 141 karaktert beolvasott. >IN-t nullára állitja.

WORD --- b

PAD-hez hozzáadja az >IN-ben levő eltolást, és az ezen cimen található bájtot a veremre teszi. Ezután >IN tartalmát eggyel növeli.

VLIST

Ugyanaz, mint a FORTH-79 VLIST-je, de minden szónév kiirása után várakozik. A következő szónevet az Y billentyű megnyomása után irja ki.

f

Megegyezik a FORTH-79 FORGET utasitásával.

WAIT

Várakozik, amig meg nem nyomják az Y billentyűt. A program futása csak ezután folytatódik.

STKSWP

A lebegőpontos FORTH mind az adat-, mind a visszatérési vermet a Z80 SP regiszterén keresztül kezeli. Az éppen használt verem mutatója az SP-ben, a másiké a 23728 cimen levő átmeneti tárolóban van. STKSWP a tároló és az SP tartalmát cseréli meg.

R@ --- n

Két bájtot másol a visszatérésiről az adatveremre. (FORTH-79 R szava).

SPtoCS

Az SP által éppen cimzett veremről másol át egy lebegőpontos számot a SPECTRUM számitási vermére. A számitási vermen az értéktelen bájt nem jelenik meg.

2toCS

Kétszer hat bájtot visz át az adatról a számitási veremre. A számitási vermen az értéktelen bájtok nem jelennek meg.

CStoD --- l

Lebegőpontos számot visz át a számitási veremről az adatveremre.

+! ." >IN ABORT CR EMIT EXIT
EXPECT FIND PAD QUIT SPACE SPACES

A FORTH-79-nél leirtakkal azonos paraméterezésű és működésű szavak, illetve változók.

F3.5. SIF-Z80 FORTH 2.2.

A SIF-Z80 FORTH 2.2. a CP/M operációs rendszer alatt futtatható FORTH, amely elsősorban vezérlési feladatok megoldására szolgál. Igen sok helyen eltér a szabvány FORTH-tól: a sok egyszerűsités mellett jelentős mennyiségű speciális szolgáltatással rendelkezik. Három beépitett adattipusa van:

- Egész számok. Ezek előjelből és számjegyekből állnak. A hexa számokat a szám után irt H (1E7AH, -E00H stb.), a bináris számokat a szám után irt B jelzi (101011B, -11001B, stb.). Csak ez a három számrendszer használható (2-es, 10-es, 16-os).

- Lebegőpontos számok. Ezek előjelből, egész- és törtrészből, valamint tizedespontból, illetve előjelből, egész- és törtrészből, tizedespontból, valamint exponensből állhatnak. (-12., 3.14159, 6.2E-3, stb.).

- Karakterfüzérek. A füzér "-k között adható meg, tetszőleges karaktereket tartalmazhat a " kivételével. A SIF-Z80 FORTH-ban szinte az összes veremkezelő és relációs műveletnek (DUP, ROT, <, = stb.) van füzérre vonatkozó megfelelője. A karakterfüzérek a vermen és a szótárban a következő módon tárolódnak:

0 A1 A2 A3 ... An (n+2)*2

Ai magas bájtja tartalmazza a füzér i-edik karakterének kódját (az alacsony bájt értéktelen), n pedig a füzér hossza.

A szabványtól a szavak belső ábrázolása is eltér. Az alapszótár szavainál nincs csatolómező, a felhasználói szótár

szavainál a csatolómező helyén egy listamutató mező van. A szavak láncolását egy, a névmezőben található eltolás segitségével valósitja meg (NFA_{i+1} = NFA_i+eltolás).

A SIF-Z80 FORTH minden szódefinició forrásszövegét is tárolja, ASCII karakteres formában. Minden felhasználói szó listamutatója saját forrásszövegére mutat. A teljes lista módositható a beépitett editorral, újrafordítható, kiirható a lemezre CP/M fájl formájában, onnan bármikor újból visszaolvasható és lefordítható. Ezzel helyettesitettük a kissé körülményes screenkezelést.

A következőkben közöljük a szabvány FORTH-tól eltérő működésű szavakat, majd pedig a beépitett speciális funkciókat. n a 16 bites egész számokat, l a lebegőpontos számokat, f a feltételt (0: hamis, 1: igaz), s pedig a füzéreket jelöli. fn olyan füzér, amely fájlnévként értelmeződik. Az ezekben előforduló * és ? karakterek a CP/M-ben megszokott jelentésűek.

.B n ---
n-et binárisan irja ki.

.H n ---
n-et hexadecimálisan irja ki.

VOCABULARY
A szótár szavainak nevét és cfa-ját irja ki. A FORTH-79 VLIST utasitásához hasonló.

VLIST

A felhasználói szótár szavait irja ki a képernyőre, definiciójukkal együtt.

LIST

< LIST szónév > hatására a szódefinició forrásszövegét irja ki.

DO ... +LOOP

A /LOOP kivételével az összes FORTH-79 feltételes és vezérlőszerkezet megtalálható, és egyedül a DO ... +LOOP szintaktikája eltérő:

nköz nvég nkezdő DO szósor +LOOP

tehát az nköz a cikluson kivül adandó meg.

Közvetlen szó, amely az őt követő szó kódmezőcimét teszi a szótárba. A cim elé egy LIT-et is fordit.

--- cim

A következő szó névmezőcimét teszi a veremre.

NFA --- nfa

Rendszerváltozó, amelyben a legutoljára definiált szó névmezőcime van.

CFA --- cfa
Rendszerváltozó, amelyben a legutoljára definiált szó kódmezőcime van.

PFA --- pfa
Rendszerváltozó, amelyben a legutoljára definiált szó paramétermező-cime van.

TYPE cim ---
Kiirja cimtől kezdve az ott található bájtokat, ASCII karakterek formájában. A kiirást egy 0 bájt megtalálásáig folytatja.

HEX n1 --- n2 n3
n2 és n3 n1 alsó két bájtja hexa számjegyének ASCII kódja lesz.

SWAB n1 --- n2
n1 két bájtját felcseréli.

^ n1 n2 --- n3
Egész hatványozás. n3=n1^n2.

TRUE --- 1
Igaz értéket szolgáltató feltételkonstans.

FALSE --- 0
Hamis értéket szolgáltató feltételkonstans.

INKEY --- n

Leolvassa a billentyűzet állapotát. Ha nincs lenyomva billentyű, akkor n=0, egyébként n a billentyű ASCII kódja.

& s ---

A vermen található füzért irja ki.

SYSLIB

Ha valamelyik szódefinició után SYSLIB-et irunk, akkor ez és az ezt megelőző szavak rendszerszóként értelmeződnek, és rájuk a FORGET, LIST, VLIST szavak hatástalanok.

RESTORE

Megszünteti a SYSLIB hatását.

LOAD fn ---

A vermen levő füzért fájlnévként értelmezve beolvassa és leforditja az adott fájlban levő szótárt.

SAVE fn ---

Hatására az eddig definiált felhasználói szótár forrás-szövegű listája a lemezre kerül.

NEW

A memóriában levő alap- és felhasználói szótárakból egy új FORTH rendszert generál, majd kiirja annak hosszát bájtokban. Ezután a BYE paranccsal visszatérhetünk a CP/M-be és annak SAVE utasitásával az új rendszert .COM kiterjesztéssel a lemezre menthetjük.

RECOMP

A RECFROM rendszerváltozóban tárolt listacimtől kezdve újrafordítja a szótárat a forrásszöveg felhasználásával.

UERROR n ---

A vezérlést a felhasználó által definiálható tetszőleges hibakezelő szóra adja. A hibakezelő szó n-et mint hibakódot használhatja fel.

ERR

A felhasználó által definiált hibakezelő szó törzsének kötelezően az ERR az első szava. Az UERROR az ERR-t tartalmazó szóra adja a vezérlést.

DIR fn ---

< "fájlnév" DIR > egyenértékű a CP/M DIR fájlnév parancsával.

ERA fn ---

< "fájlnév" ERA > egyenértékű a CP/M ERA fájlnév parancsával.

INOPEN fn ---

Hatására a FORTH az adott fájlt szekvenciális olvasásra megnyitja.

OUTOPEN fn ---

Hatására a FORTH az adott fájlt szekvenciális írásra megnyitja.

INCLOSE

A FORTH lezárja az olvasásra megnyitott fájlt.

OUTCLOSE

A FORTH lezárja az irásra megnyitott fájlt.

READ --- n

Beolvas egy bájtot az olvasásra megnnyitott fájlból, és azt a veremre teszi. n felső bájtja nulla lesz.

WRITE n ---

n alsó bájtját az irásra megnyitott fájlba irja.

IM0

A Z80 0-ás megszakitásmódját állitja be.

IM1

A Z80 1-es megszakitásmódját állitja be.

IM2

A Z80 2-es megszakitásmódját állitja be.

HALT

Hatására a processzor egy HALT utasitást hajt végre és megszakitásra várakozik.

CODE cim ---

Szubrutinként meghivja a cimen kezdődő gépi kódú programot. A rutinból RET utasitással kell visszatérni.

CALL cim AF BC DE HL --- AF BC DE HL

Meghivja a cimen kezdődő gépi kódú szubrutint a Z80 regisztereinek megfelelő sorrendű feltöltésével. A rutinból való visszatérés után a regiszterek tartalma a veremre kerül a fenti sorrendben.

IT

A szódefinició törzsében elsőnek szereplő IT szó hatására maszkolható megszakitás esetén az igy definiált szóra kerül a vezérlés.

NMI

A szódefinició törzsében elsőnek szereplő NMI szó hatására nem maszkolható megszakitás esetén az igy definiált szóra kerül a vezérlés.

ITNMI!

Végrehajtása után az eddigi megszakitásrutinok helyett az IT-t, ill. NMI-t tartalmazó szavak fogják elvégezni a megszakitáskezelést.

EI

Engedélyezi a maszkolható megszakitást.

DI

Letiltja a maszkolható megszakitást.

IN n1 --- n2

Az n1 portcimről beolvas egy bájtot és a veremre teszi. n2 felső bájtja nulla lesz.

OUT n1 n2 ---

Az n1 alsó bájtja kiiródik az n2 portcimre.

BMOV 0 n1 n2 ... ni cim --- nb

Cimtől kezdődően lerakja az ni-től a nulláig a számok magas bájtját. A letett bájtok számát (nb) a vermen visszaadja.

A további szolgáltatások .FTH kiterjesztésű fájlokban találhatók. Igy ezeket csak akkor kell beolvasni, ha szükség van rájuk, egyébként nem foglalják el feleslegesen a memóriát. Ezeket a lemezen tárolt szótárakat csak vázlatosan ismertetjük. A két assemblert nem számitva összességükben körülbelül 200 új szóval bővitik az alapszótárat.

REAL.FTH

Ez tartalmazza a teljes lebegőpontos aritmetikát:

- veremkezelő szavak,
- aritmetikai szavak,
- relációk,
- memóriakezelés,
- lebegőpontos változók és konstansok definiálása,
- függvények.

STRING.FTH

Olyan szavak definicióit tartalmazza, melyek paraméterei füzérek:

- veremkezelő szavak (DUP$, ROT$, stb.),
- relációk,
- függvények (LEFT$, RIGHT$, stb.),
- füzérváltozók és konstansok,
- memóriakezelés (@$, !$),
- füzérek egyesitése (+$).

ARRAY.FTH, ARRAYX.FTH, ARRAY$.FTH

Egész, lebegőpontos és karakterfüzér-vektorok és mátrixok definiálását teszi lehetővé.

CURSOR.FTH

A fájl a mindenkori terminál kurzorvezérlő szavait (CLEAR, HOME, UP, stb.) és kódjait tartalmazza. A képernyőeditornak például ez az egyetlen display-függő szótára.

EDIT.FTH

Egy képernyőorientált editort tartalmaz, amellyel szavainkat javithatjuk és módosithatjuk. Az < EDIT szónév > utasitás hatására a képernyő törlődik és megjelenik az adott szó definiciójának szövege. Ezt szerkeszthetjük a kurzormozgató és DEL billentyűkkel, valamint tetszőleges beszúrásokat alkalmazhatunk. A szerkesztés befejeztével a FORTH újrafordítja a szótárt az adott szótól kezdődően.

TKB.FTH

Ez a SIF-Z80 FORTH task buildere. Segitségével a FORTH-ban megirt szótárakat egy közvetlenül (a FORTH

rendszer behivása nélkül) futtatható taskká szerkeszthetjük össze. A szerkesztés kétféle lehet:

- a CP/M operációs rendszer alatt önállóan futtatható programot hoz létre, amely egy 100H kezdőcimű bináris fájl;

- INTEL-HEX formátumú ASCII fájlt generál. A ROM és a RAM kiosztását a felhasználó tetszőlegesen választhatja meg. Ezzel egy bármilyen konfigurációjú Z80 rendszeren futtatható program jön létre, amely szükség esetén akár ROM-ba is égethető.

Mindkét esetben a TKB eltávolitja a szótárból a végrehajtáshoz felesleges információkat (névmező és listamutatómező), valamint kihagyja az alapszótárból az összes olyan szót, amelyek a FORTH program futásához szükségtelenek. Az igy keletkező, csak szótörzsekből álló programot ezután a szerkesztéstől függően 100h-ra vagy a felhasználó által megadott cimre relokálja. A leszerkesztett program az eredeti szótárnál lényegesen rövidebb lesz.

I80.FTH; Z80.FTH

A két fájl egy-egy assemblert tartalmaz.

DIS.FTH

Ebben a fájlban egy FORTH "decompiler" található, amely lehetővé teszi akár az alap-, akár a felhasználói szótár szavainak visszafejtését szótárbeli törzsük alapján.

F4. A FORTH szavak betűrendben

Az alábbi táblázat a FORTH szavakat foglalja össze betűrendben. Az egyes szavaknál feltüntetjük, hogy melyik fejezet (pont) tárgyalja, valamint, hogy mely - általunk ismertetett - FORTH implementációban található meg ("x" jelöli ha igen, "-" ha nem). A rövidítések a következők:

fig = fig-FORTH
CPM = 8080 CPM FORTH 2.2
C64 = Commodore C-64 FORTH
SPA = Spectrum ABERSOFT FORTH
SPF = ZX SPECTRUM 48k Floating Point FORTH
SIF = SIF-Z80 FORTH 2.2

Utasitás	pont	fig	CPM	C64	SPA	SPF	SIF
!	2.3.1.1	x	x	x	x	x	x
!CSP	6.8.	x	x	x	x	-	x
#	2.7.2.1	x	x	x	x	-	x
#>	2.7.2.1	x	x	x	x	-	-
#S	2.7.2.1	x	x	x	x	-	-
#BUFF	6.7.2	x	x	x	x	-	-
'	6.5.2	x	x	x	x	-	x
(	3.1.1	x	x	x	x	-	x
)	3.1.1	x	x	x	x	-	x
(.")	6.8	x	x	x	x	-	-
(;CODE)	6.8	x	x	x	x	-	-
(+LOOP)	6.8	x	x	x	x	-	x
(ABORT)	6.8	x	x	x	x	-	-
(DO)	6.8	x	x	x	x	-	x
(FIND)	6.5.2	x	x	x	x	-	-
(LINE)	5.3.3.5	x	x	x	x	-	-

F4. A FORTH szavak betűrendben

Utasítás	pont	fig	CPM	C64	SPA	SPF	SIF
(LOOP)	6.8	x	x	x	x	-	x
(NUMBER)	2.7.2.2	x	x	x	x	-	-
*	2.5.1	x	x	x	x	x	x
*/	2.5.1	x	x	x	x	-	-
*/MOD	2.5.1	x	x	x	x	-	-
+	2.5.1	x	x	x	x	x	x
+!	2.3.1.1	x	x	x	x	x	x
+-	2.5.1	x	x	x	x	-	-
+BUF	6.8	x	x	x	x	-	-
+LOOP	3.3.2.1	x	x	x	x	x	x
+ORIGIN	6.8	x	x	x	x	-	-
-	2.5.1	x	x	x	x	x	x
-->	5.2.3.1	x	x	x	x	-	-
-DUP	2.4.1	x	x	x	x	-	-
-FIND	6.5.2	x	x	x	x	-	-
-TEXT	2.8	-	-	-	-	-	-
-TRAILING	2.8	x	x	x	x	x	-
,	5.1.2	x	x	x	x	-	x
"	2.9.2	x	x	x	x	x	x
.	2.9.1	x	x	x	x	x	x
."	2.9.2	x	x	x	x	x	-
.CPU	6.8	x	x	-	x	-	-
.LINE	5.3.3.5	x	x	x	x	-	-
.R	2.9.1	x	x	x	x	-	-
/	2.5.1	x	x	x	x	x	x
/LOOP	3.3.2.1	-	-	-	-	-	-
/MOD	2.5.1	x	x	x	x	-	-
0 1 2 3	6.7.2	x	x	x	x	-	-
0<	2.6.1	x	x	x	x	x	x
0=	2.6.1	x	x	x	x	x	x
0>	2.6.1	-	-	-	-	x	x
0BRANCH	6.6	x	x	x	x	-	x

F4. A FORTH szavak betűrendben

Utasítás	pont	fig	CPM	C64	SPA	SPF	SIF
1+	2.5.1	x	x	x	x	x	x
1-	2.5.1	-	-	x	-	x	x
2+	2.5.1	x	x	x	x	x	x
2-	2.5.1	-	-	x	-	x	x
2*	2.5.1	-	-	x	-	-	-
2/	2.5.1	-	-	-	-	-	-
2!	2.3.1.1	x	x	x	x	-	-
2@	2.3.1.1	x	x	x	x	-	-
2CONSTANT	2.3.2	-	-	x	x	-	-
2DROP	2.4.1	-	-	x	x	-	-
2DUP	2.4.1	x	x	x	x	-	-
2OVER	2.4.1	-	-	x	x	-	-
2SWAP	2.4.1	-	-	x	x	-	-
2VARIABLE	2.3.3	-	-	x	x	-	-
:	3.1.1	x	x	x	x	-	x
;	3.1.1	x	x	x	x	-	x
;CODE	5.4.3.2	x	x	x	x	-	-
;S	5.3.3.1	x	x	x	x	-	x
<	2.6.1	x	x	x	x	x	x
<#	2.7.2.1	x	x	x	x	-	-
<BUILDS	5.2.1	x	x	x	x	-	x
=	2.6.1	x	x	x	x	x	x
>	2.6.1	x	x	x	x	x	x
>BINARY	2.7.2.2	-	-	-	-	-	-
>IN	6.7.1	-	-	-	-	x	-
>R	2.4.2	x	x	x	x	x	x
?	2.9.1	x	x	x	x	x	-
?COMP	6.6.1	x	x	x	x	-	-
?CSP	6.8	x	x	x	x	-	-
?DUP	2.4.1	-	-	-	-	x	-
?ERROR	6.8	x	x	x	x	-	x
?EXEC	6.6.1	x	x	x	x	-	-

F4. A FORTH szavak betűrendben

Utasítás	pont	fig	CPM	C64	SPA	SPF	SIF
?LOADING	6.8	x	x	x	x	-	-
?PAIRS	6.6.1	x	x	x	x	-	x
?STACK	6.8	x	x	x	x	-	x
?TERMINAL	6.8	x	x	x	x	-	-
@	2.3.1.1	x	x	x	x	x	x
ABORT	3.6.1	x	x	x	x	x	x
ABS	2.5.1	x	x	x	x	x	-
AGAIN	3.3.2.5	x	x	x	x	-	x
ALLOT	5.1.1	x	x	x	x	-	x
AND	2.6.2	x	x	x	x	x	x
ASSEMBLER	5.4.3.2	-	-	x	-	-	x
B/BUF	5.3	x	x	x	x	-	-
	6.7.2	x	x	x	x	-	-
B/SCR	5.3.2	x	x	x	x	-	-
	6.7.2	x	x	x	x	-	-
BACK	6.6.1	x	x	x	x	-	x
BASE	2.7.2	x	x	x	x	-	-
	6.7.1	x	x	x	x	-	-
BEGIN	3.3.2.3	x	x	x	x	x	x
	3.3.2.4	x	x	x	x	x	x
	3.3.2.5	x	x	x	x	x	x
BL	2.7.2.1	x	x	x	x	-	-
	6.7.2	x	x	x	x	-	-
BLANK	2.3.1.2	-	-	-	-	-	-
BLANKS	2.3.1.2	x	x	x	x	-	-
BLK	6.7.1	x	x	x	x	-	-
BLOCK	5.3.1.1	x	x	x	x	-	-
BRANCH	6.6	x	x	x	x	-	x
BUFFER	5.3.1.2	x	x	x	x	-	-
BYE	6.8	x	x	-	x	-	x
C/L	6.7.2	x	x	x	x	-	-
C!	2.3.1.1	x	x	x	x	x	x

F4. A FORTH szavak betűrendben

Utasítás	pont	fig	CPM	C64	SPA	SPF	SIF
C,	5.1.2	x	x	x	x	-	x
C;	5.4.3.2	-	-	-	-	-	-
C@	2.3.1.1	x	x	x	x	x	x
CASE	4.3	-	-	-	x	-	x
CFA	6.5.2	x	x	x	x	-	x
CMOVE	2.3.1.2	x	x	x	x	x	x
CODE	5.4.3.2	-	-	x	-	-	x
COLD	3.6.3	x	x	x	x	-	x
COMPILE	3.5.1	x	x	x	x	-	x
CONSTANT	2.3.2	x	x	x	x	-	x
CONTEXT	5.4	x	x	x	x	-	-
	6.7.1	x	x	x	x	-	-
CONVERT	2.7.2.2	-	-	-	-	-	-
COPY	5.3.1.6	-	x	x	-	-	-
COUNT	2.8	x	x	x	x	x	-
CR	2.9.2	x	x	x	x	x	x
CREATE	5.2.1	x	x	x	x	-	-
CSP	6.7.1	x	x	x	x	-	x
CURRENT	5.4	x	x	x	-	-	-
	6.7.1	x	x	x	x	-	-
D0=	2.6.1	-	-	-	-	-	-
D<	2.6.1	-	-	-	-	-	-
D=	2.6.1	-	-	-	-	-	-
D+	2.5.2	x	x	x	x	-	-
D-	2.5.2	-	-	-	-	-	-
D+-	2.5.2	x	x	x	x	-	-
D.	2.9.1	x	x	x	x	-	-
D.R	2.9.1	x	x	x	x	-	-
DABS	2.5.2	x	x	x	x	-	-
DECIMAL	2.7.2	x	x	x	x	-	-
DEFINITIONS	5.4.2	x	x	x	x	-	-
DEPTH	2.4.1	-	-	-	-	x	x

F4. A FORTH szavak betűrendben

Utasítás	pont	fig	CPM	C64	SPA	SPF	SIF
DIGIT	2.7.3	x	x	x	x	-	-
DLITERAL	3.5.4	x	x	x	x	-	-
DMAX	2.5.2	-	-	-	-	-	-
DMIN	2.5.2	-	-	-	-	-	-
DMINUS	2.5.2	x	x	x	x	-	-
DNEGATE	2.5.2	-	-	-	-	-	-
DO	3.3.2.1	x	x	x	x	x	x
DOES>	5.2.1	x	x	x	x	-	x
DP	3.2.1.2	x	x	x	x	-	x
	6.7.1	x	x	x	x	-	x
DPL	2.7.2.2	x	x	x	x	-	-
	6.7.1	x	x	x	x	-	x
DROP	2.4.1	x	x	x	x	x	x
DU<	2.6.1	-	-	-	-	-	-
DUMP	2.3.1.2	-	-	x	-	x	x
DUP	2.4.1	x	x	x	x	x	x
EDITOR	5.4.3.3	x	x	x	x	-	-
ELSE	3.3.1.1	x	x	x	x	x	x
EMIT	2.9.2	x	x	x	x	x	x
EMPTY-BUFFERS	5.3.1.5	x	x	x	x	-	-
ENCLOSE	2.8	x	x	x	x	-	-
END	3.3.2.4	x	x	x	x	-	x
ENDCASE	4.3	-	-	-	x	-	x
ENDIF	3.3.1.1	x	x	x	x	x	x
ENDOF	4.3	-	-	-	x	-	x
ERASE	2.3.1.2	x	x	x	x	x	-
ERROR	6.8	x	x	x	x	-	x
EXECUTE	6.6.2	x	x	x	x	x	x
EXPECT	2.9.3	x	x	x	x	x	-
FENCE	3.2.1.2	x	x	x	x	-	x
	6.7.1	x	x	x	x	-	x
FILL	2.3.1.2	x	x	x	x	x	-

F4. A FORTH szavak betűrendben

Utasítás	pont	fig	CPM	C64	SPA	SPF	SIF
FIND	6.5.2	-	-	-	-	x	x
FIRST	6.7.2	x	x	x	x	-	-
FLD	6.7.1	x	x	x	x	-	-
FLUSH	5.3.1.4	x	x	x	x	-	-
FORGET	3.2.1.2	x	x	x	x	x	x
FORTH	5.4.3.1	x	x	x	x	-	-
HERE	3.2.1.2	x	x	x	x	-	x
HEX	2.7.2	x	x	x	x	-	x
HLD	6.7.1	x	x	x	x	-	-
HOLD	2.7.2.1	x	x	x	x	-	-
I	2.4.2	x	x	x	x	x	x
	3.3.2.2	x	x	x	x	x	x
I'	2.4.2	x	x	x	x	-	-
	3.3.2.2	x	x	x	x	-	x
ID.	6.5.2	x	x	x	x	-	-
IF	3.3.1.1	x	x	x	x	x	x
IMMEDIATE	3.4	x	x	x	x	-	x
IN	6.7.1	x	x	x	x	-	x
INDEX	5.3.3.3	x	x	x	x	-	-
INTERPRET	6.6.3	x	x	x	x	-	x
J	2.4.2	-	-	x	x	x	-
	3.3.2.2	-	-	x	x	x	-
KEY	2.9.3	x	x	x	x	x	x
LATEST	6.5.2	x	x	x	x	-	-
LEAVE	3.3.2.2	x	x	x	x	x	x
LFA	6.5.2	x	x	x	x	-	-
LIMIT	6.7.2	x	x	x	x	-	-
LIST	5.3.3.2	x	x	x	x	-	x
LIT	3.5.4	x	x	x	x	-	x
LITERAL	3.5.4	x	x	x	x	-	x
LOAD	5.3.3.1	x	x	x	x	-	x
LOOP	3.3.2.1	x	x	x	x	x	x

F4. A FORTH szavak betűrendben

Utasítás	pont	fig	CPM	C64	SPA	SPF	SIF
M+	2.5.3	-	-	-	-	-	-
M*	2.5.3	x	x	x	x	-	-
M*/	2.5.3	-	-	x	-	-	-
M/	2.5.3	x	x	x	x	-	-
M/MOD	2.5.3	x	x	x	x	-	-
MAX	2.5.1	x	x	x	x	x	x
MESSAGE	6.8	x	x	x	x	-	-
MIN	2.5.1	x	x	x	x	x	x
MINUS	2.5.1	x	x	x	x	-	x
MOD	2.5.1	x	x	x	x	-	x
MYSELF	3.3.3.2	-	-	-	-	-	x
MOVE	2.3.1.2	-	-	-	-	-	x
NEXT	5.4.3.2	x	x	x	x	-	x
	6.4.1	x	x	x	x	-	x
NEGATE	2.5.1	-	-	-	-	x	-
NFA	6.5.2	x	x	x	x	-	x
NOOP	6.8	x	x	-	x	-	-
NOT	2.6.2	-	-	-	x	x	x
NUMBER	2.7.2.2	x	x	x	x	-	-
OF	4.3	-	-	-	x	-	x
OR	2.6.2	x	x	x	x	x	x
OUT	6.7.1	x	x	x	x	-	x
OVER	2.4.1	x	x	x	x	x	x
PAD	2.2.2.2	x	x	x	x	x	-
PAGE	2.9.2	-	-	x	-	x	-
PFA	6.5.2	x	x	x	x	x	x
PICK	2.4.1	-	-	x	-	x	x
PREV	6.7.1	x	x	x	x	-	x
QUERY	2.9.3	x	x	x	x	x	-
QUIT	3.6.2	x	x	x	x	x	x
R	2.4.2	x	x	x	x	-	x
R/W	5.3.1.8	x	x	x	x	-	-

F4. A FORTH szavak betűrendben

Utasítás	pont	fig	CPM	C64	SPA	SPF	SIF
R>	2.4.2	x	x	x	x	x	x
R@	2.4.2	x	x	-	x	-	x
	6.7.1	x	x	-	x	-	x
R0	2.4.2	-	-	-	-	x	-
REPEAT	3.3.2.3	x	x	x	x	x	x
ROLL	2.4.1	-	-	-	-	x	x
ROT	2.4.1	x	x	x	x	x	x
RP!	2.4.2	x	x	x	x	-	x
	3.6.1	x	x	x	x	-	x
RP@	2.4.2	x	x	-	x	-	x
S->D	2.7.1	x	x	x	x	-	-
S0	2.4.1	x	x	-	x	x	x
	2.8	x	x	-	x	x	x
	6.7.1	x	x	-	x	x	x
SAVE-BUFFERS	5.3.1.4	-	-	-	-	-	-
SCR	5.3	x	x	x	x	-	-
	6.7.1	x	x	x	x	-	-
SIGN	2.7.2.1	x	x	x	x	-	-
SMUDGE	3.3.1.1	x	x	x	x	-	-
SP!	2.4.1	x	x	x	x	-	x
	3.6.1	x	x	x	x	-	x
SP@	2.4.1	x	x	x	x	-	x
SPACE	2.9.2	x	x	x	x	x	x
SPACES	2.9.2	x	x	x	x	x	x
STATE	6.7.1	x	x	x	x	-	x
SWAP	2.4.1	x	x	x	x	x	x
TASK	6.8	x	x	-	x	-	-
TEXT	2.8	-	-	x	x	-	-
THEN	3.3.1.1	x	x	x	x	x	x
TIB	2.4.1	x	x	x	x	-	x
	2.8	x	x	x	x	-	x
	6.7.1	x	x	x	x	-	x

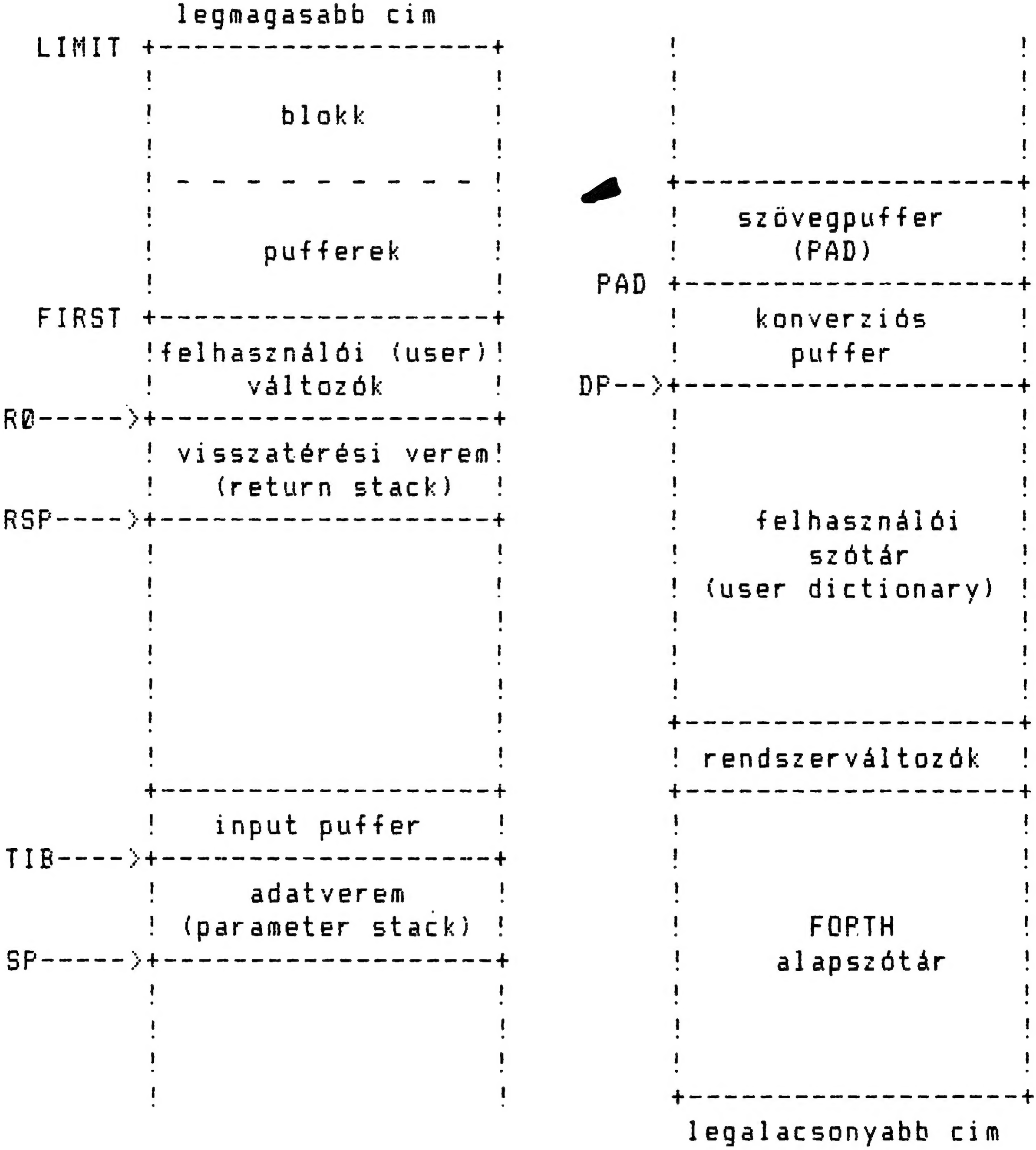
F4. A FORTH szavak betűrendben

Utasitás	pont	fig	CPM	C64	SPA	SPF	SIF
TOGGLE	2.3.1.1	x	x	x	x	-	-
TRAVERSE	6.5.2	x	x	x	x	-	-
TRIAD	5.3.3.4	x	x	x	x	-	-
TYPE	2.9.2	x	x	x	x	x	x
U.	2.9.1	x	x	x	x	-	-
U.R	2.9.1	-	-	-	x	-	-
U*	2.5.3	x	x	x	x	-	-
U/	2.5.3	x	x	x	-	-	-
U<	2.6.1	x	x	x	x	-	-
UNTIL	3.3.2.4	x	x	x	x	x	x
UPDATE	5.3.1.3	x	x	x	x	-	-
USE	6.7.1	x	x	x	x	-	-
USER	2.3.3	x	x	x	x	-	x
VARIABLE	2.3.3	x	x	x	x	-	x
VLIST	3.2.1.2	x	x	x	x	-	x
VOC-LINK	6.7.1	x	x	x	x	-	-
VOCABULARY	5.4.1	x	x	x	x	-	x
WARM	3.6.3	x	x	-	-	-	x
WARNING	6.7.1	x	x	x	x	-	-
WHILE	3.3.2.3	x	x	x	x	x	x
WIDTH	6.7.1	x	x	x	x	-	-
WIPE	5.3.1.7	-	x	-	x	-	-
WORD	2.8	x	x	x	x	x	-
XOR	2.6.2	x	x	x	x	-	x
[	3.5.3	x	x	x	x	-	x
[COMPILE]	3.5.2	x	x	x	x	-	x
]	3.5.3	x	x	x	x	-	x

F5. FORTH-79 definicók

```
SCR # 1
  0 ( FORTH-79 STANDARD definiciók )
  1
  2 VOCABULARY 79-STANDARD IMMEDIATE
  3 79-STANDARD DEFINITIONS DECIMAL
  4 : VARIABLE 0 VARIABLE ;
  5 : 2VARIABLE VARIABLE 2 ALLOT ;
  6 : CONVERT (NUMBER) ;
  7 : >IN IN ;
  8 : ?DUP -DUP ;
  9 : CREATE VARIABLE -2 ALLOT ;
 10 : SAVE-BUFFERS FLUSH ;
 11 : NEGATE MINUS ;
 12 : DNEGATE DMINUS ;
 13 : 0> MINUS 0< ;
 14 : DEPTH SP@ S0 @ SWAP - 2 / ;
 15 -->
SCR # 2
  0 ( FORTH-79 STANDARD definiciók, folytatás )
  1
  2 : WORD WORD HERE ;
  3 : MOVE 2 * CMOVE ;
  4 : U/MOD U/ ;
  5 : D< ROT 2DUP = IF ROT ROT DMINUS D+ 0< ELSE SWAP <
  6    SWAP DROP ENDIF SWAP DROP ;
  7 : R@ R ;
  8 : NOT 0= ;
  9 : PICK DUP + SP@ + @ ;
 10 : ROLL DEPTH >R DUP DUP R> < >R 1 > R> AND IF 1+ DUP
 11    PICK SWAP DUP + SP@ + 2 - SP@ 1+ SWAP DO I @ I 2 + !
 12    -2 +LOOP DROP ELSE DROP ENDIF ;
 13 : I' R> R> R@ SWAP >R SWAP >R ;
 14 : J R> R> R> R@ SWAP >R SWAP >R SWAP >R ;
 15 FORTH DEFINITIONS DECIMAL ;S
```

F6. A FORTH rendszer memóriaképe



F7. Irodalomjegyzék

[1] Leo Brodie: Starting FORTH.
Prentice-Hall, Englewood Cliffs, 1981.

[2] FORTH Interest Group: FORTH-79 Standard.
San Carlos, California, 1979.

[3] S.D. Roberts: FORTH Applications.
ELCOMP Publishing, Inc., 1985.

[4] Lipovszki-Subai-Beszeda: FORTH.
LSI ATSZ, 1985.

[5] The FORTH language.
Wireless World, 1983.

[6] FORTH DIMENSIONS, 1983.

[7] ABERSOFT FORTH.
Melbourne House, 1984.

[8] Mike Hampson: SPECTRUM FLOATING-POINT FORTH.
Hampson and Lord, 1983.

[9] Seres-Fenyő-Gyalogh-Boér: SIF-Z80 FORTH 2.2 Felhasználói leirás.
MTA Központi Fizikai Kutatóintézet, 1985.

[10] C.H. TING, PhD: SYSTEMS GUIDE TO fig-FORTH.
OFFETE ENTERPRISES, Inc.

[11] PET-FORTH, datatronic ab.

Kiadja a Műszaki Könyvkiadó
Felelős kiadó: Fischer Herbert igazgató
Felelős vezető: Horváth Józsefné dr.
Révai Nyomda, Eger – 84 3183 10000

Műszaki vezető: Kőrizs Károly
Műszaki szerkesztő: Mózer István
A kötést tervezte: Kováts Tibor
A könyv ábráit rajzolták a szerzők
A könyv formátuma: B5
Ívterjedelme: 26.5 (A5)
Ábrák száma: 33
Papír minősége: 65 g. fam. reg. nyomó
Azonossági szám: 61 411
MÜ: 3806–i–8688
Készült az MSZ 5601 és 5602 szerint
A kézirat lezárva: 1985. december